Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado em Engenharia Informática
2º Semestre, 2009/2010

Modelação de *Concerns* Voláteis em aplicações geoespaciais: O caso dos *Concerns* Geoespaciais

N.º 30225, Sara Patrícia Rovisco Farto Machado Silva

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Informática

Orientador
Prof. Doutor João Baptista da Silva Araújo Júnior

Co-Orientadora
Profª. Doutora Maria Armanda Simenta Rodrigues Grueau

Lisboa

(2010)

Nº do aluno: 30225

Nome: Sara Patrícia Rovisco Farto Machado Silva

Título da dissertação:

Modelação de *Concerns* Voláteis em aplicações geoespaciais: O caso dos *Concerns* Geoespaciais

Palavras-Chave:

- Sistemas de Informação Geográfica (SIG)
- Engenharia de Requisitos
- Análise de Domínio
- Padrões de Análise
- Desenvolvimento de *Software* Orientado a Aspectos

Keywords:

- *Geographical Information System (GIS)*
- *Requirements Engineering*
- *Domain Analysis*
- *Analysis Patterns*
- *Aspect-Oriented Software Development*

Agradecimentos

Agradeço a todas as pessoas que de alguma forma me apoiaram, directa ou indirectamente, ao longo desta dissertação.

Aos meus orientadores, o professor João Araújo e a professora Armanda Rodrigues, pela paciência e disponibilidade que demonstraram ao longo deste percurso de preparação e elaboração da dissertação. Foi muito bom poder contar com o vosso apoio sempre presente, mesmo nas piores alturas.

Ao professor Vasco Amaral, pela ajuda que me deu na implementação da DSL e na avaliação da ferramenta produzida.

Ao Vasco Sousa, por partilhar o seu conhecimento comigo, pela sua preciosa contribuição, disponibilidade e paciência para me ajudar na implementação da DSL proposta nesta dissertação. Aos colegas, que se disponibilizaram para participar no processo de avaliação desta dissertação.

À minha companheira Rita por todo o apoio e companheirismo que demonstrou ao longo destes últimos anos. Bem como a todos os meus colegas que trabalharam comigo diariamente, tornando a elaboração desta dissertação muito mais fácil e divertida. Ao meu grande amigo e namorado, por todo o apoio, carinho, amizade e felicidade que me proporcionou ao longo destes anos.

Finalmente, os meus pais por terem feito de mim aquilo que sou hoje, por todo o amor, carinho e compreensão que me deram ao longo destes 24 anos. À minha família e amigos por todo o apoio, carinho e amizade que me deram, mesmo eu não estando tão presente quanto desejava.

Obrigada a todos!

Resumo

O desenvolvimento de *Interfaces* de Programação de Aplicações (do inglês *Application Programming Interface* ou *API*) de mapas para Internet e a disponibilidade de dispositivos de posicionamento global têm conduzido a um aumento da disponibilidade e utilização dos Sistemas de Informação Geográfica (SIG) *online*. Nestes sistemas, alterar dados e funcionalidades levanta problemas de eficiência, pois as funcionalidades adicionadas podem ser voláteis, como numa aplicação de cálculo de rotas, a alteração de circulação devido a uma emergência em determinada área, pode conduzir à necessidade de alterar rapidamente a implementação do respectivo algoritmo. Isto representa um problema em aplicações SIG, onde a utilização de ferramentas de modelação é frequentemente limitada à estruturação dos dados. Assim, outras perspectivas de modelação (e.g., comportamental) são necessárias para especificar e representar estes assuntos (do inglês *concerns*) espaciais voláteis (que apresentam padrões de comportamento) em SIG, de forma modularizada e reutilizável. O Desenvolvimento de *Software* Orientado a Aspectos (DSOA) tem sido utilizado com sucesso para obter modelos mais modularizados e reutilizáveis, inclusive para modelar aplicações SIG. Esta dissertação insere-se no contexto do projecto *Aspect-Web*. O seu objectivo é obter SIGs mais aptos a lidar com a mudança ao nível da análise, recorrendo à reutilização de especificações. Assim, propomos a identificação e descrição de padrões de análise, recorrendo a mecanismos DSOA na modelação; e uma ferramenta para descrever estes padrões.

Abstract

The development of Maps Application Programming Interfaces – API – for the Internet and the availability of Global Positioning Devices have led to increased availability of online Geographic Information Systems (GIS). In these systems, to change data and functionality raises problems of efficiency, because the added features can be volatile, as in a route finding application, the change of traffic routing, due to an emergency in one area, may lead to the need for quickly changing the implementation of routing algorithm. This represents a problem in GIS applications, where the use of modeling tools is often limited to the data structure. Thus, other perspectives of modelling (e.g., behavioral) are required to specify and represent these GIS volatile spatial concerns (which show behavioral patterns) in a modularized and reusable way. Aspect-Oriented Software Development (AOSD) has been used successful in obtaining more modularized and reusable models, including int the modeling of GIS applications. This work fits in the context of the Aspect-Web project. Its goal is to enable GIS applications to deal with change at the analysis level, through reuse of specifications. Therefore, we propose the identification and description of analysis patterns, using AOSD mechanisms in modeling, and a tool to describe these patterns.

Lista de Acrónimos

SIG	**S**istemas de **I**nformação **G**eográfica
GIS	**G**eographic **I**nformation **S**ystems
API	**A**pplication **P**rogramming **I**nterface
GRICES	**G**abinete de **R**elações **I**nternacionais da **C**iência e do **E**nsino **S**uperior
FCT/UNL	**F**aculdade de **C**iências e **T**ecnologia/**U**niversidade **N**ova de **L**isboa
LIFIA	**L**aboratorio de **I**nvestigación y **F**ormación en **I**nformática **A**vanzada
DSOA	**D**esenvolvimento de **S**oftware **O**rientado a **A**spectos
CAD	**C**omputer-**A**ided **D**esign
IG	**I**nformação **G**eográfica
PDA	**P**ersonal **D**igital **A**ssistants
HTTP	**H**ypertext **T**ransfer **P**rotocol
HTML	**H**yper**T**ext **M**arkup **L**anguage
JPG (JPEG)	**J**oint **P**hotographic **E**xperts **G**roup
GIF	**G**raphics **I**nterchange **F**ormat
PNG	**P**ortable **N**etwork **G**raphics
AJAX	**A**synchronous **J**avascript **A**nd **X**ML
XHTML	e**X**tensible **H**yper**T**ext **M**arkup **L**anguage
AOSD	**A**spect **O**riented **S**oftware **D**evelopment
CASE	**C**omputer-**A**ided **S**oftware **E**ngineering
LDE	**L**inguagem de **D**omínio **E**specífico
DSL	**D**omain **S**pecific **L**anguage

RF	**R**equisito **F**uncional
RNF	**R**equisito **N**ão-**F**uncional
LPG	**L**inguagem de **P**rogramação de propósito **G**eral
GPL	**G**eneral-purpose **P**rogramming **L**anguage
MDD	**M**odel **D**riven **D**evelopment
MOF	**M**eta **O**bject **F**acility
OMG	**O**bject **M**anagement **G**roup
UML	**U**nified **M**odeling **L**anguage
OCL	**O**bject **C**onstraint **L**anguage
EMF	**E**clipse **M**odeling **F**ramework
GEF	**G**raphical **E**diting **F**ramework
GMF	**G**raphical **M**odeling **F**ramework

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1. Contexto

Esta dissertação está a ser desenvolvida no âmbito do projecto *Aspect-Web: Developing Web Applications with Aspects*, financiado por um acordo bilateral entre o Gabinete de Relações Internacionais da Ciência e do Ensino Superior de Portugal (GRICES) e a Secretaria de Estado da Ciência, Tecnologia e Inovação Produtiva da Argentina (SECyT) (Oliveira, 2009).

Neste projecto participam a Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (FCT/UNL) e o Laboratório de Investigação e Formação em Informática Avançada (Lifia) da *Universidad Nacional de La Plata*, Argentina.

Com este projecto pretende-se investigar o desenvolvimento de mecanismos para modelar de forma mais modularizada Sistemas de Informação Geográfica (SIG), na fase de especificação de requisitos e de desenho. Este projecto pretende promover a modularidade e consequentemente a evolução dos SIG.

1.2. Motivação

Os sistemas georreferenciados *online*, com origem nos Sistemas de Informação Geográfica (SIG) têm vindo a ganhar um papel cada vez mais activo na sociedade. Com a difusão dos SIG através da *Web*, devido à grande disponibilidade dos Dispositivos de Posicionamento Global e ao desenvolvimento de *APIs* de mapas para usar na internet, este comportamento veio intensificar-se ainda mais. Como exemplos de sistemas que beneficiariam da georreferenciação temos o CLIP (o sistema de inscrições e de informação das disciplinas e horários dos alunos, docentes e todos os outros colaboradores da Universidade Nova de Lisboa), onde se poderia mostrar a localização das salas, aulas, docentes, alunos, etc. É de notar que este sistema será usado nesta dissertação como caso de estudo.

As aplicações SIG são usadas nas mais diversas áreas, como aplicações ambientais, de planeamento do território, de transportes, governamentais, de controlo de propagação de doenças, entre outras.

Como acontece com todas as outras aplicações, a sua concepção deve ser iniciada com uma fase de análise de requisitos. Nesta fase, deve reunir-se toda a informação sobre a aplicação, sobre as suas funcionalidades, sobre os seus utilizadores, sobre o seu âmbito, os seus riscos, sobre possíveis futuras aplicações, entre outras. A análise de requisitos em SIG é fundamental, dado o peso dos dados neste tipo de aplicações, na estruturação e concepção do suporte de informação. A modelação comportamental, neste tipo de aplicações é, de forma geral, baseada num conjunto de operadores considerados fundamentais pela comunidade envolvida, e disponibilizados de uma forma genérica.

No entanto, as aplicações geoespaciais *online* que conhecem um grande crescimento hoje em dia envolvem propriedades inerentemente voláteis, implicando uma necessidade considerável de manutenção e actualização, não só das estruturas de informação, mas também da componente comportamental. Algumas dessas propriedades voláteis também podem ser recorrentes em várias aplicações. Uma vez identificadas, deveria ser possível especificá-las de tal forma que fossem facilmente reutilizáveis.

Por outro lado, a falta de modularização poderá levantar problemas de flexibilidade e reutilização. Além disso, este tipo de manutenção tem de ser levado a cabo com pleno conhecimento do domínio em que se está a trabalhar e com recurso às devidas técnicas de apoio à manutenção. Para se conhecer o domínio em que se está a trabalhar, deve ser feita uma análise prévia do mesmo.

A análise de domínio consiste na identificação, análise e especificação dos requisitos comuns pertencentes a uma aplicação de domínio específico, para que sejam reutilizados na especificação e construção de outros projectos de software que venham a ser criados (Arango, 1989; Prieto-Díaz, 1990; Pressman, 2005).

Com o processo da análise de domínio, pretende-se que toda a informação de desenvolvimento fique disponível, para que, quando for necessário decidir entre reutilizar, ou não, determinado componente, o engenheiro de *software* possa aceder à totalidade da informação que lhe permita compreender todo o contexto que levou o *designer*, criador do componente, a fazê-lo de determinada forma. Assim, o engenheiro de *software*, além de poder tomar uma decisão consciente e informada, pode ainda tornar a reutilização mais eficaz (Prieto-Díaz, 1990).

Assim, com a análise de domínio será possível ter um conhecimento profundo do domínio, tornando-se simples perceber o que numa aplicação pode ser reutilizado em outras aplicações do mesmo domínio. Nomeadamente, tornar-se-á mais fácil a identificação de possíveis padrões de análise.

Os padrões de análise são especificações usadas para proceder à reutilização. Um padrão é uma especificação (e solução) de um problema que pode ser reutilizada em diversos contextos (Hamza *et* al., 2004). Os padrões de análise surgem para ajudar na construção de modelos de análise, por conta da sua reutilização, sendo as suas principais preocupações os modelos conceptuais, a flexibilidade e a capacidade de reutilização dos módulos de sistemas que resultam da aplicação destes padrões.

Estes padrões facilitam o desenvolvimento de modelos de análise de onde se obtêm os principais requisitos do sistema, proporcionando modelos de análise reutilizáveis, e facilitam a transformação destes modelos em modelos de *design* (Pressman, 2005).

Assim sendo, torna-se clara a vantagem existente em obter uma abordagem para modelação de Sistemas de Informação Geográfica, partindo de uma boa análise de domínio, para posteriormente poder obter os padrões de análise que possam vir a permitir a reutilização de partes de um sistema.

Para além do bom conhecimento do domínio, dada a característica volátil de alguns dos assuntos (do inglês *concerns*, a ser usado daqui por diante) espaciais, é fundamental que haja um bom controlo dos mesmos, nomeadamente, é muito importante que estes sejam identificados e que os *concerns* relacionados com estes sejam também identificados. Os referidos *concerns* são voláteis no sentido em que podem ser requeridos num determinado momento, deixando mais tarde de ser necessários, e podem ser considerados transversais, na medida em que podem ser considerados em vários módulos das aplicações.

O Desenvolvimento de *Software* Orientado a Aspectos caracteriza-se por permitir a identificação, modularização e composição dos *concerns* transversais (do inglês *crosscutting concerns*) no início do ciclo de vida do *software* (Oliveira, 2009). Um *concern* é qualquer assunto de interesse num sistema de *software* (Sutton *et* al., 2005). Um *concern* diz-se ser *crosscutting* (transversal) se estiver *tangled* (embaraçado) com outro *concern* num único módulo ou se está *scattered* (disperso) por diversos módulos do sistema (Figueiredo *et* al., 2005).

Segundo Sommerville (2007), a grande vantagem das abordagens orientadas a aspectos é promover uma melhor separação dos *concerns*. A separação dos *concerns* em elementos independentes, em vez de incluir diferentes *concerns* na mesma abstracção lógica, é uma boa prática de engenharia de *software*. Representando os *crosscutting concerns* como aspectos, estes podem ser compreendidos, reutilizados e modificados de forma independente. A modelação de *concerns* voláteis utilizando aspectos foi descrita por Moreira *et* al. (2006).

Figueiredo *et* al. (2005), consideram que, além da separação dos *concerns*, o DSOA tem ainda as vantagens de minimizar a replicação de código, melhorando a coesão entre os diversos módulos, de reduzir o entrelaço do sistema e consequente aumento da possibilidade de reutilização, e facilidade de evolução no desenvolvimento de sistemas de *software* complexos. Além disso utilizar os mecanismos de modelação e composição de DSOA para representação e composição de padrões foi algo considerado e utilizado neste trabalho, devido ao sucesso verificado quando aplicado em algumas abordagens (Araújo *et* al., 2004).

Assim, a motivação para a elaboração desta dissertação é a criação de uma abordagem orientada a aspectos para modelação de *concerns* voláteis em SIG, mais concretamente dos *concerns* geoespaciais.

1.3. Objectivos

Os objectivos propostos para esta dissertação têm em vista tornar a implementação dos SIG mais eficiente. Para atingir este objectivo é necessário:

- Que se tenha disponível uma boa análise do domínio da mesma;
- Que se identifiquem os *concerns* voláteis em SIG que se repetem em mais do que uma aplicação;
- Que se identifiquem os padrões de análise existentes no domínio;
- Que se definam esses padrões usando um *template* adequado ao domínio geoespacial, para que estes possam constituir um catálogo que permitirá a reutilização de módulos e funcionalidades das aplicações relevantes, podendo vir a ser usado para outras aplicações do mesmo âmbito.

Foi também objectivo desta dissertação, a criação de uma ferramenta de suporte à criação de padrões, em modo computacional. Esta ferramenta deveria integrar todas as etapas de descrição do padrão. Ou seja, desejava-se que existisse uma ferramenta capaz de permitir a

criação de Diagramas de *Features*, de Classes e de Sequência de forma integrada no restante *template* de descrição.

1.4. Contribuições

As principais contribuições desta dissertação são:

- Descrição dos padrões identificados;
- Inserção do Diagrama de *Features* na descrição do padrão;
- Introdução dos mecanismos de Desenvolvimento de Software Orientado a Aspectos na modelação do padrão;
- Ferramenta *PatternTool* que serve de suporte computacional para a criação de padrões.

1.5. Organização do documento

Este documento está organizado do seguinte modo:

- No Capítulo 2, é apresentada a informação obtida da pesquisa bibliográfica relativa aos Sistemas de Informação Geográfica (SIG) e sobre *Web* SIG. Este capítulo está dividido em 4 partes. Uma parte inicial em que é explicado o que são SIGs e *Web* SIGs e a sua evolução. Uma segunda em que se apresentam os tipos de *Web* SIG e se focam as suas diferenças. Uma terceira, onde são apresentadas e explicadas as arquitecturas *Web* SIG. Um quarta onde se aborda o tema das *Maps API*, mais concretamente o caso do *Google Maps*. E finalmente, uma quinta onde se fala da modelação de SIG;

- No Capítulo 3, é apresentada a pesquisa bibliográfica sobre Engenharia de Requisitos e Reutilização. Inicialmente, são explicados os conceitos principais de requisitos e sua classificação. Neste Capítulo, apresentam-se ainda as técnicas de reutilização de especificações a serem usadas em SIG (i.e., análise de domínio, padrões de análise e modelação com aspectos);

- No Capítulo 4, são apresentados alguns trabalhos desenvolvidos em áreas desta dissertação;

- No Capítulo 5, é apresentado o template de padrão que é usado na descrição dos mesmos. São, também, apresentadas as descrições dos dois padrões desenvolvidos nesta dissertação;

- No Capítulo 6, são apresentados alguns conceitos sobre metamodelação. É também apresentada a ferramenta desenvolvida, bem como todo o processo que foi usado para a criação dos editores que a compõem;
- No Capítulo 7, são apresentados os resultados da avaliação, isto é, são apresentadas e analisadas as questões constantes nos questionários;
- No Capítulo 8, são apresentadas as conclusões desta dissertação, nomeadamente as vantagens e desvantagens do trabalho desenvolvido, as limitações do trabalho e finalmente o trabalho que poderá ser feito no futuro.

Esta dissertação possui ainda 7 Anexos:

- Anexo A – Questionário 1: Avaliação da Ferramenta;
- Anexo B – Questionário 2: Avaliação do Padrão;
- Anexo C – Metamodelo *emf* do Editor Principal;
- Anexo D – Metamodelo *emf* do Editor do Diagrama de *Features;*
- Anexo E – Metamodelo *emf* do Editor do Diagrama de Classes;
- Anexo F – Metamodelo *emf* do Editor do Diagrama de Sequência;
- Anexo G – Manual de Utilização da Ferramenta *PatternTool.*

2. Sistemas de Informação Geográfica (SIG)

Os Sistemas de Informação Geográfica (SIG) surgiram no início dos anos 70 e tiveram um peso preponderante no desenvolvimento da análise geográfica e na fixação da ideia de que a geografia é uma área de estudo, essencialmente espacial (Dragicevic, 2004).

Os SIG são ferramentas tecnológicas que facilitam a compreensão do espaço geográfico e a tomada de decisões de forma inteligente. Estes sistemas organizam dados geográficos para que o utilizador, através da sua visualização integrada numa base geográfica, possa gerar resultados que lhe permitam compreender o problema sobre o qual se encontra a trabalhar (ESRI, 2008).

Segundo Dragicevic (2004), um SIG é um sistema computacional que suporta o uso e tratamento de dados sobre o espaço geográfico. São essencialmente sistemas de informação compostos por *software*, *hardware,* dados espaciais e operações computacionais, que têm como objectivo recolher, modelar, armazenar, partilhar, recuperar, manipular e mostrar dados referenciados geograficamente.

Os SIG evoluíram dos pacotes *CAD – Computer-Aided Design* ou Desenho Assistido por Computador –, dos sistemas de apresentação de mapas e de *software* de base de dados, e de *spreadsheets*, sendo a maior vantagem dos SIG perante estes, a integração de todos eles, permitindo ao utilizador proceder a uma análise espacial complexa que com os anteriores dificilmente se conseguiria (Dana, 1993).

Segundo Dragicevic (2004), apenas nos anos 90 se começou a investigar a possibilidade de disponibilizar e manipular informação geográfica na Internet. Assim, em 1993, a *Xerox Corporation* desenvolveu a primeira aplicação experimental de visualização de mapas para pesquisa interactiva de dados espaciais na *Web*. Em 1994, o projecto *Alexandria Digital Library*, financiado pela *US National Science Foundation,* lançou o primeiro serviço bibliotecário distribuído para dados referenciados espacialmente. Estas primeiras iniciativas para integrar a Internet com as tecnologias SIG prenderam-se, essencialmente, com a

divulgação de mapas estáticos, seguindo-se os mapas interactivos com suporte para *design* Cliente/Servidor e ferramentas avançadas para cartografia e geovisualização.

As aplicações SIG baseadas na *Web* evoluíram para produzir Serviços Distribuídos de Informação Geográfica (IG) para a Internet (*Internet Distributed GIServices)* com capacidade para interagir com sistemas múltiplos e heterogéneos, e servidores que suportam outras funções SIG avançadas (Dragicevic, 2004).

2.1. Tipos de *Web*-SIG

Segundo Kou-gen *et* al. (2000), e relativamente aos tipos de *Web* SIG, o investimento em *Web* SIG foi fundamentado pelo crescente interesse dos utilizadores pela *Web*. No entanto, surgiram alguns problemas, em primeiro lugar relacionados com a disponibilização dos dados espaciais em formato digital, e depois com o acesso aos dados.

A *Web* veio permitir a instalação de um servidor *Web* para produzir mapas e gerar gráficos através dos dados fornecidos pelo utilizador, sendo possível actualizar estes mapas e gráficos, em tempo real, com novos dados que vão sendo recebidos. Estas actualizações ficam ainda disponíveis para todos os utilizadores ao mesmo tempo, pois os dados espaciais são facilmente acessíveis em qualquer ponto do globo, através da *Web*, permitindo também a sua criação e alteração a partir de qualquer localização.

Os *Web* SIG oferecem diversas capacidades, como algumas das seleccionadas por Kou-gen:

- Ligação entre um ficheiro de imagem e uma localização geográfica;
- Ligação de vários tipos de *media* a posições/localizações geográficas;
- Configuração de mapas temáticos pelos utilizadores;
- Integração de dados SIG com imagens de satélite para referenciação visual do "mundo real";
- Análise espacial (embora limitada);
- Anotações *online* sobre a informação espacial;
- Comércio electrónico;
- Actualizações de dados *online*;
- Apoio à decisão;
- Acesso aos dados em tempo real.

Os *Web* SIG aproveitaram todas as vantagens conferidas aos sistemas distribuídos e, potencialmente, têm muitas outras adicionais, como as apresentadas abaixo (Kou-gen *et* al., 2000):

- É possível aceder em qualquer altura ao servidor *Web* SIG, qualquer que seja a sua localização e, desde que este esteja disponível;
- O acesso simultâneo aos dados permanentemente actualizados dos mais variados servidores *Web* SIG localizados por todo o mundo, de forma fácil;
- Simplicidade e familiaridade do *Web browser* de acesso a servidores *Web* SIG por todo o mundo;
- O facto de ser apenas necessário ter uma ligação à Internet e um *Web browser*, para aceder e manipular a informação geográfica de um servidor *Web* SIG;
- A capacidade de analisar interactivamente um *layer* geográfico;
- A possibilidade de organizar os mapas de acordo com o desempenho do sistema;
- A hipótese dos utilizadores poderem construir o seu próprio mapa;
- O facto das bases de dados poderem ser consultadas através da *interface* de um mapa;
- A possibilidade de distribuir directamente através de um *site*, a informação que outrora era distribuída "em mão", para outras entidades ou para o público em geral.

Assim, Kraak (2004) identificou algumas vantagens relativamente aos mapas na *Web*, que são comuns a todos os mapas digitais, como a possibilidade de fazer *zoom-in* ou *zoom-out*, de fazer *pan*, de visualizar um *layer* ou não simplesmente com um *click* (ligar/desligar a visualização de um *layer*). Existe ainda a capacidade de ter acesso à base de dados existente por detrás do mapa, bastando para isso clicar num símbolo. É também possível levar a cabo operações simples como medir distâncias ou áreas.

No seu trabalho, Kraak (2004) assume que, actualmente, a *Web* é o meio para apresentar e difundir mapas e dados geoespaciais. A *Web* permite oferecer as plataformas de mapas e de dados de forma independente e um número ilimitado de utilizadores pode aceder virtualmente aos mapas de qualquer sítio e a qualquer hora. No entanto, ainda existem muitos problemas relacionados com a *Web* que se reflectem nos *Web* SIG. O desempenho do trabalho num

ambiente *Web* SIG depende de diversos factores tecnológicos, como a conexão à Internet, a intensidade do tráfego, a eficiência dos dados, e a capacidade das máquinas cliente e servidor.

Kraak (2004) refere ainda que as outras desvantagens dos *Web* SIG estão relacionadas com aspectos práticos de usabilidade. Os mapas de papel oferecem uma melhor usabilidade no campo, embora o *hardware* recente como os PDA ou *tablet's* avançados ofereçam alternativas interessantes. Adicionalmente, a resolução do ecrã dos diversos dispositivos pequenos irá colocar restrições extras no *design*. Ainda mais crítico é que, hoje em dia, o utilizador espera que a informação esteja actualizada. No passado, a manutenção dos mapas era muito mais suportada por cartógrafos que tinham em conta as necessidades dos utilizadores. Actualmente, justamente por isso, é preciso ter em conta uma exigência dirigida ao ambiente de utilização e algumas das necessidades cartográficas podem ser resolvidas por algoritmos usados enquanto se criam os *mapas conforme as necessidades.

Kou-gen *et* al. (2000) dividem os *Web* SIG em 3 categorias, de acordo com o momento em que os documentos geográficos são gerados, descritas a seguir.

***Web* SIG Estático.** O documento geográfico do *Web* SIG Estático é guardado como um ficheiro e o seu criador fixa o seu conteúdo. Assim, qualquer pedido de documento de mapa estático tem sempre a mesma resposta – Figura 2.1.

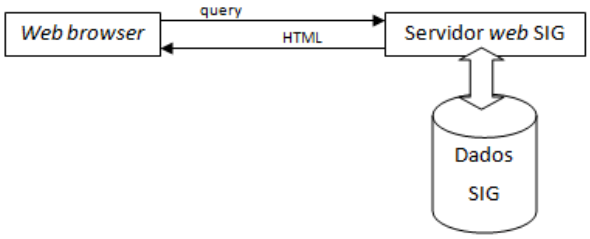

Figura 2.1. *Web* SIG Estático (Kou-gen *et* al., 2000)

As principais vantagens deste tipo de *Web* SIG são:

- Simplicidade na criação do documento, podendo ser criado por alguém que não saiba programar.
- Confiabilidade, pois a partir do momento em que se criam e verificam cuidadosamente as ligações, estas permanecem sempre válidas.

- Desempenho, pois é possível apresentar o mapa rapidamente, desde que o *Web browser* guarde uma cópia do documento do mapa estático em *cache* no disco local.

A principal desvantagem deste tipo é a falta de flexibilidade, pois sempre que a informação muda, é necessário editar o ficheiro manualmente. Este facto dificulta a tarefa de fornecer informação aos utilizadores de forma dinâmica.

***Web* SIG Dinâmico.** O documento geográfico do *Web* SIG dinâmico não é guardado como um ficheiro, é antes criado por um servidor de dados espaciais sempre que um *browser* pede o documento. Como sempre que é feito um pedido, é criado um novo documento geográfico com a especificidade do pedido, a resposta proporcionada por um *Web* SIG dinâmico varia de pedido para pedido – Figura 2.2.

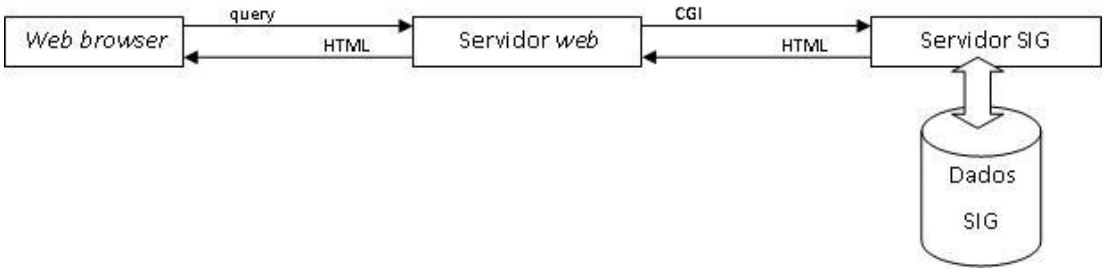

Figura 2.2.*Web*-SIG Dinâmico (Kou-gen *et* al., 2000)

A principal vantagem deste tipo de *Web* SIG é a sua capacidade de gerar documentos geográficos de forma dinâmica. No entanto, também tem desvantagens, como o maior custo em *hardware*, devido à necessidade de servidores *Web* mais potentes para responder aos pedidos de vários utilizadores ao mesmo tempo.

***Web* SIG Activo.** O documento geográfico do *Web* SIG Activo é especificado por um programa executado no *browser* da máquina local, com o qual o utilizador pode interagir, alterando continuamente o que vê, e podendo usar os dados geográficos no servidor. Quando um *browser* pede um documento de mapa de um servidor *Web* SIG activo, o servidor devolve uma cópia do programa e posteriormente o *browser* corre-o localmente. Assim, com um *Web* SIG activo, pode-se fornecer, obter, actualizar, consultar e analisar os dados geográficos no servidor.

Mais concretamente, num Web SIG activo o servidor transfere para o cliente parte do programa que lhe permitirá tratar pedidos. No entanto, quando os pedidos são demasiado

complicados, de tal forma que o programa Cliente não consegue resolvê-los, será pedido ao Servidor *Web* SIG que o resolva, e todos os resultados são devolvidos ao cliente na forma de um vector de dados (Jing-zhong *et* al., 2008).

Este tipo de *Web* SIG tem algumas vantagens, nomeadamente:

- Permite aos utilizadores visualizar e analisar dados geográficos de forma mais contínua e interactiva do que os *Web* SIG Dinâmicos.
- Em alguns casos as operações são demasiado simples sendo desnecessário enviar pedidos ao servidor, pois o programa que corre no cliente é perfeitamente capaz de as concretizar. Isto diminui demasiado o fluxo de dados entre o utilizador e o servidor, o que leva a uma maior rapidez na resposta (Jing-zhong *et* al., 2008).

Desta forma, pode-se dizer que o *Web* SIG Activo tem vantagens sobre o *Web* SIG Dinâmico.

2.2. Arquitectura dos *Web* SIG

Nos últimos tempos, têm sido aplicados, dois principais tipos de arquitectura (Luqun *et* al., 2004) no desenvolvimento de aplicações SIG baseadas na *Web:* as chamadas *two-tier* (Figura 2.3) e as *three-tier* (Figura 2.4).

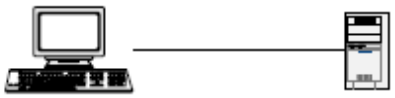

Figura 2.3. Arquitectura Cliente/Servidor *2-Tier* (Luqun *et* al., 2004)

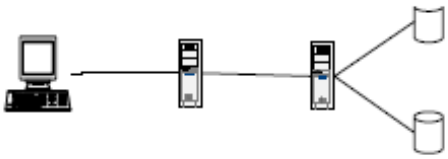

Figura 2.4. Arquitectura Cliente/Servidor *3-Tier* (Luqun *et* al., 2004)

De acordo com Luqun *et* al. (2004) a arquitectura do tipo *two-tier* é composta por *software* no cliente e *software* no servidor, sendo que há transmissão de dados espaciais do servidor para o cliente, através de um protocolo de comunicação *Web* (e.g. HTTP). Neste tipo de arquitectura o *software* do lado do cliente permite visualizar dados espaciais, e o servidor apenas fornece o serviço de base de dados.

Relativamente aos recursos existentes no lado do cliente e no lado do servidor, podemos ter dois tipos diferentes de configuração (Marshall, 2000): a configuração *client-side* (Figura 2.5) e a *server-side* (Figura 2.6).

Marshall (2000) descreve a configuração *client-side*, como podemos ver na Figura 2.5, como uma arquitectura em que o cliente é modelado de forma a fornecer o suporte necessário às funcionalidades SIG. Assim, deverá ter, como o servidor, uma base de dados e todo o suporte SIG, acessível através da *Web browser*.

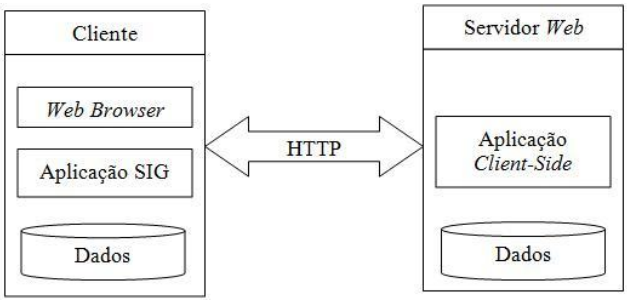

Figura 2.5. Arquitectura *Client-Side* (Marshall, 2000)

No que diz respeito à configuração *server-side* (Marshall, 2000), o cliente irá apenas fazer pedidos ao servidor e apresentar os resultados, funcionando apenas como *Web browser*. Desta forma, como podemos ver na Figura 2.6, nestas aplicações, além dos dados espaciais, todo o *software* complexo e registado, fica no servidor. O utilizador não tem, assim, acesso a muitas funcionalidades locais.

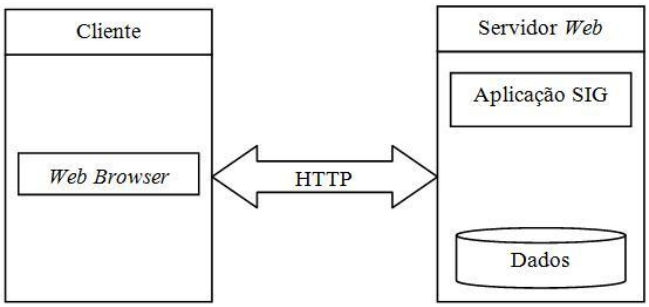

Figura 2.6. Arquitectura *Server-Side* (Marshall, 2000)

Esta arquitectura apresenta diversas vantagens, dado que tanto a aplicação como os dados estão centralizados no servidor. Tais vantagens incluem a simplicidade nas fases de desenvolvimento, aplicação e manutenção da aplicação. No entanto, a *interface*

disponibilizada ao utilizador torna-se menos flexível e apelativa, tendo este de passar parte do seu tempo à espera do resultado enviado pelo servidor.

A arquitectura *two-tier* é facilmente configurável de raiz, mas no que respeita a actualizá-la e estendê-la, a configuração torna-se mais complicada, uma vez que a ligação cliente/servidor é tipicamente baseada num protocolo privado (Luqun *et* al., 2004).

No que diz respeito à arquitectura *three-tier* (Luqun *et* al., 2004), esta identifica uma divisão da lógica de funcionamento em 3 níveis, como se pode ver na Figura 2.7:

1. O primeiro nível é a **interface do utilizador** que permite, ao mesmo tempo, a interacção com o sistema;

2. O segundo nível é o **nível de lógica de negócio** do processamento de transacções SIG, permitindo a realização de toda a lógica subjacente à aplicação em causa e à funcionalidade de acesso a dados;

3. O terceiro nível é o **nível de serviço de armazenamento de dados espaciais SIG**, que oferece a funcionalidade de armazenamento contínuo dos dados.

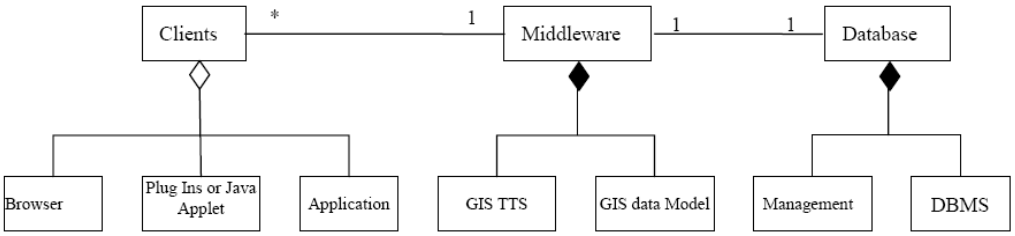

Figura 2.7. Arquitectura *Web* SIG Cliente/Servidor *3-Tier* (Luqun *et* al., 2004)

Para uma aplicação SIG baseada na *Web*, o ponto principal do *design* e da implementação é a construção de uma boa arquitectura de *software* e como dividir o módulo das funções, principalmente em relação ao nível intermédio.

Com vista a reduzir o custo do *software*, seria conveniente que houvesse do lado cliente uma redistribuição e um equilíbrio na conexão entre este e o servidor. Assim, a maioria dos programas do lado cliente são feitos em Java e incluídos numa página *Web*. A parte central no modelo *three-tier* – o *Middleware* – oferece serviços para os dados SIG, isolando localmente

o lado cliente do nível dos dados, para reduzir a complexidade do acesso aos dados no lado cliente, e ao mesmo tempo aumentar a segurança da base de dados.

Nesta arquitectura, os programas do lado cliente são executados por *downloads* dinâmicos do *Web browser*. Assim, o lado cliente deve efectuar o *download* do *software* para cada operação *Web* SIG, e este não pode ser usado se o *browser* estiver no estado *offline*.

De acordo com Luqun *et* al. (2004) "Actualmente, todas as arquitecturas de *Web* SIGs mais maduras adoptam a arquitectura cliente/servidor *three-tier"*.

Chow (2008) descreve a arquitectura conceptual de uma aplicação *Web* de uma forma bastante simples, usando *Maps API* – Figura 2.8. De forma geral, a aplicação *Web* é alojada num servidor *Web* que vai retornar ficheiros HTML (formato *Hypertext Markup Language*) e gráficos compatíveis com a *Web* (JPG, GIF, PNG) mediante um pedido de um *Web browser*. Usando *JavaScript* associado a uma *API* de mapas (*Maps API)* disponibilizada por um fornecedor de serviços geográficos *online* (e.g. Google Maps API), a aplicação *Web* tem acesso aos servidores do mesmo fornecedor através de pedidos (serviços) disponibilizados pela *API*, como *zoom-in* e *zoom-out*. Baseado nos parâmetros de *input* e nos valores recolhidos pela *interface* do mapa da aplicação *Web*, o servidor *Web* do fornecedor vai devolver os dados espaciais (ou seja, o mapa) num formato gráfico compatível com a *Web*.

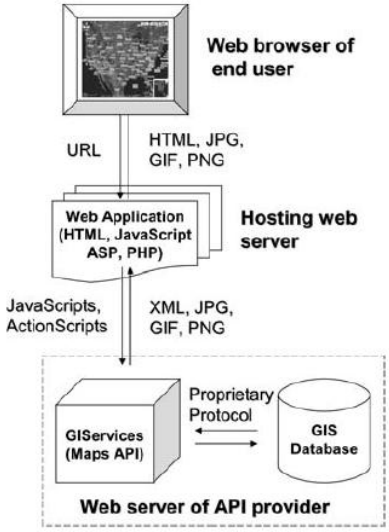

Figura 2.8. Arquitectura do sistema de um *site Web* simples que usa as funcionalidades *built-in* e dados fornecidos pela *Maps API* (Chow, 2008)

Segundo Chow (2008) as *Maps API* são *source code interfaces* que garantem aos programadores o acesso às bibliotecas de programas e que permitem solicitar serviços para a geração de mapas através da Internet. Estas terão surgido de grandes servidores de mapas na *Web* que forneceram uma cobertura extensiva dos dados espaciais por todo o mundo.

O sucesso das *Maps APIs* deve-se, em grande parte, à sua política *freeware*, à disponibilidade da camada de dados, à especificação aberta, à facilidade de implementação de aplicações adicionais com base na mesma, à dinâmica de navegação e à capacidade de consulta.

2.3. *Maps API* – O caso da *API* do *Google Maps*

Detalha-se em seguida o módulo *GIService* (*Maps API*) utilizando o exemplo da *API* do *Google Maps*.

A *Google* forneceu aos utilizadores da Internet duas aplicações SIG: o *Google Maps* – Figura 2.9 – e o *Google Earth* – Figura 2.10. Ambas as aplicações podem ser usadas por qualquer pessoa e sem qualquer custo, desde que possua acesso à Internet. Nestas condições, estas aplicações oferecem a qualquer utilizador, diversas imagens e outras informações georreferenciadas de todo o planeta, permitindo aos utilizadores visitar qualquer sítio do mundo virtualmente, bem como ver os pontos de interesse existentes em determinado local, como restaurantes, hospitais, escolas, monumentos, museus, etc. Actualmente, em algumas zonas, existe ainda a possibilidade de usar os mesmos no modo *Street View*, dando ao utilizador a sensação de estar a "passear" nas ruas.

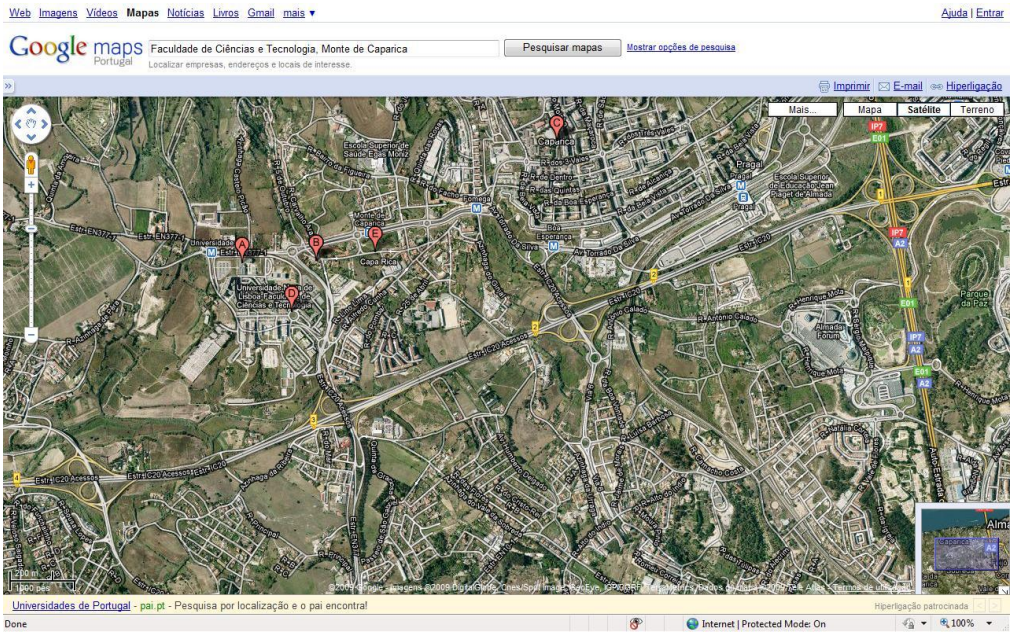

Figura 2.9. *Google Maps* - Vista de Satélite

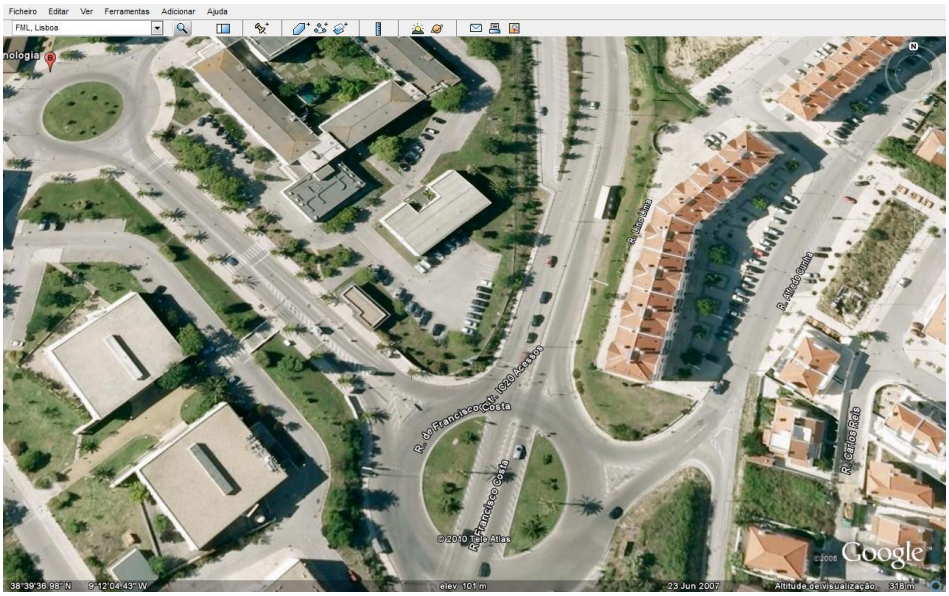

Figura 2.10. Google Earth

Segundo Pejic *et* al. (2009) o *Google Maps* é um serviço de mapeamento baseado na *Web* fornecido pela *Google*, que proporciona uma *interface* visual extremamente simples de compreender, construída usando as tecnologias AJAX. Este serviço tem detalhado os dados das ruas e das imagens aéreas, e fornece uma *API* que permite a personalização do mapa de *output*, incluindo a capacidade de adicionar dados específicos da aplicação ao mapa, e ainda permite integrar resultados acessíveis de outras instituições.

A *Google* disponibilizou uma *API* para desenvolvimento de aplicações suportadas pela base geográfica do *Google Maps*, que consiste numa *interface* gratuita que permite aos utilizadores terem acesso, nos seus próprios *sites*, a mapas disponibilizados pela *Google*, integrados com os seus próprios dados. Isto permitiu que muitas novas aplicações geográficas surgissem. Exemplos disto é o guia da cidade[1] que disponibiliza uma aplicação *online*, no seu *site*, que permite aceder a informação georreferenciada relativamente a determinada zona, calcular caminhos, etc. – Figura 2.11.

[1] http://www.guiadacidade.pt/portugal/

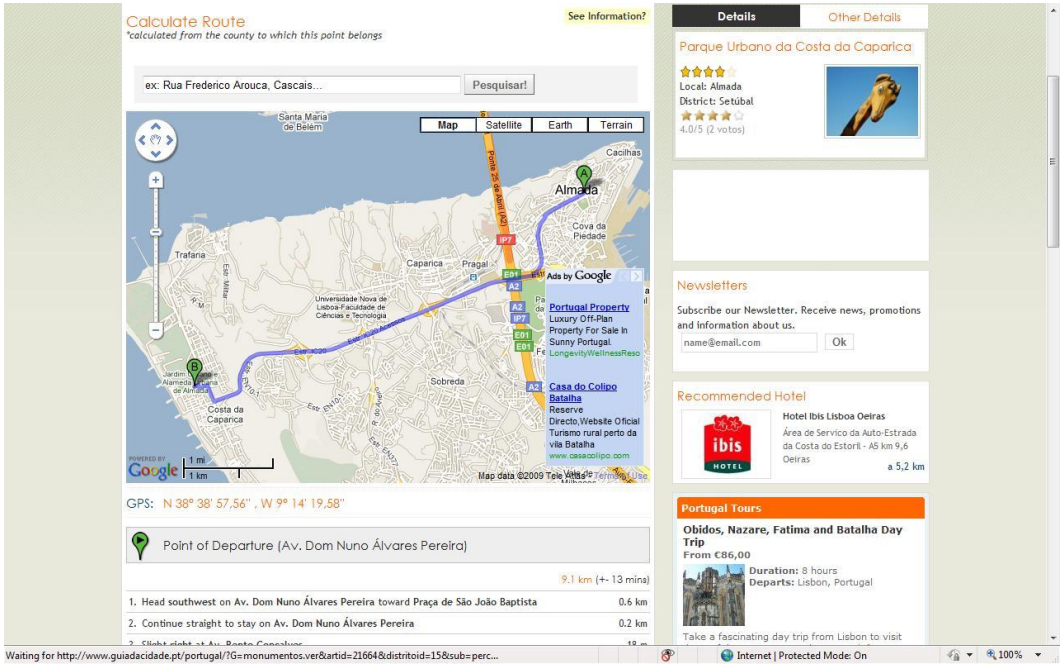

Figura 2.11. Aplicação de localização de um ponto de interesse em determinada zona e cálculo do caminho até lá no site Guia da Cidade

A *API* do *Google Maps* é baseada num conjunto de classes (componentes) de simples utilização, através de um *container* de JavaScript incorporado directamente numa página XHTML. Estas classes são carregadas do *Google* cada vez que abrimos uma página *Web* do *Google Maps*. Todas as funcionalidades do *Google Maps* são baseadas neste *container* JavaScript incorporado, construído dentro de uma página *Web*. Este componente fornece a *interface* ao serviço *Google Maps* e gera o mapa nos ecrãs dos utilizadores carregando os componentes necessários de imagem e compondo-os convenientemente no ecrã. A *interface* da *API* esconde algumas classes, objectos e *interfaces* poderosas que permitem manipular o *Google Maps*. Combinando esta *interface* com os dados que queremos mostrar, torna-se possível suportar elementos interactivos na página *Web* sem ter de recarregar a página ou recolocar as porções do mapa no ecrã – todo o processo é possível através de JavaScript conectado com o servidor *Google Maps*.

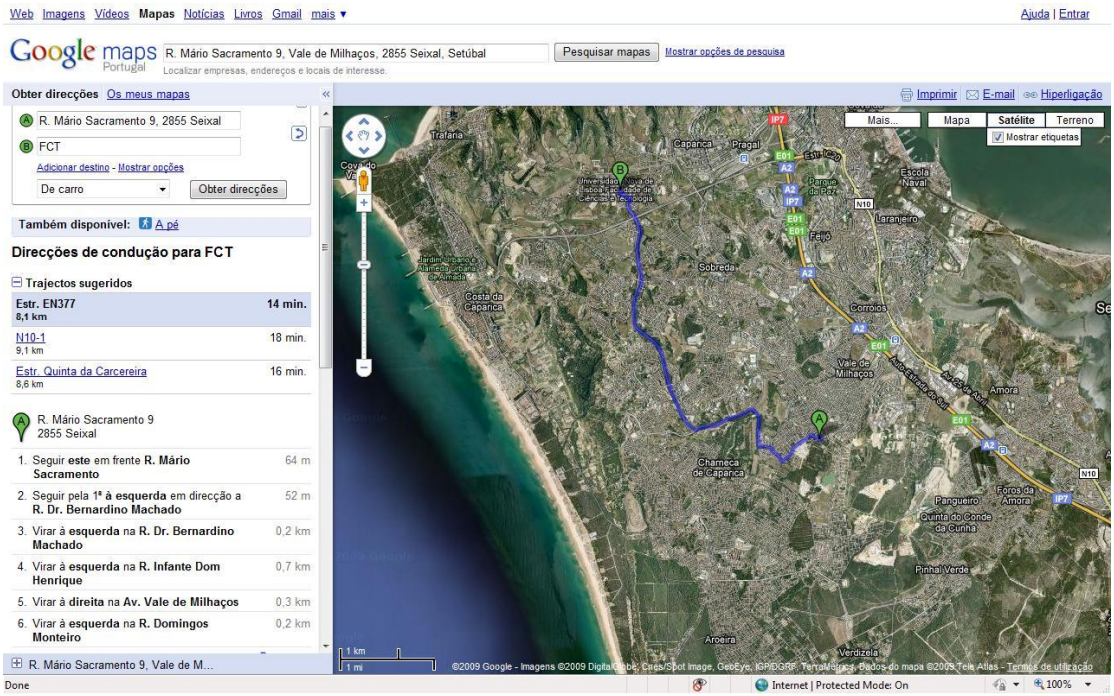

Figura 2.12. Ilustração do pedido de direcções no *Google Maps* para ir do ponto A para o ponto B

A *API* do *Google Maps* é essencialmente uma *interface* para acesso a um conjunto de classes usadas num *script* JavaScript, para construir aplicações dentro dos limites da página *Web*. Geralmente, as aplicações *Google Maps* começam com um simples mapa. A base cartográfica disponibilizada pela *Google*, e delimitada pela janela utilizada na aplicação, constitui o conteúdo do mapa original. A este mapa acrescentam-se diversos elementos disponibilizados pelas funcionalidades da *API*. Peijic *et* al. (2009) descrevem as potencialidades de alguns destes elementos:

- **Sobreposições**, utilizadas para indicar os pontos de interesse no mapa ou linhas que limitam áreas, caminhos ou outras informações relativas a localizações que têm de ser destacadas. Pode-se observar uma marcação no mapa da Figura 2.12, onde está o caminho a percorrer marcado com uma linha azul;

- **Eventos**, que são despoletados por determinadas acções, como o utilizador clicar, ou passar com o rato sobre um ponto de interesse, entre outras;

- **Controlos**, são os controlos básicos da interface que permitem ao utilizador fazer *zoom-in* e *zoom-out* no mapa, mover o mapa (*panning*), mudar o tipo de mapa (Satélite, Terreno e Mapa);

- **Serviços**, os serviços estendem a *API* do *Google Maps* adicionando novas características e funcionalidades. A *API* do *Google Maps* fornece diversos serviços, como:

  - ***Geocoding*** que, de um endereço dado, fornece a sua localização no mapa. Este serviço está também explícito na Figura 2.12, onde está a identificação das localizações A e B no mapa;

  - ***Directions*** que disponibiliza a descrição visual e textual de um caminho, de um ponto do mapa para outro, com instruções detalhadas, incluindo o cálculo da distância e do tempo que demora ir de um ponto ao outro. Mais uma vez, este serviço está também mostrado na Figura 2.12, onde estão as direcções que conduzem do ponto A ao ponto B;

  - ***Streetview*** que disponibiliza uma imagem virtual do local, exactamente como se uma pessoa lá estivesse. Pode-se observar um exemplo de *streetview* na Figura 2.13;

  - ***GoogleBar*** para fazer uma pesquisa local do mapa. Pode-se ver a *GoogleBar*, por exemplo na Figura 2.12, onde se pesquisa pelo ponto A.

Figura 2.13. *StreetView* no *Google Maps*

A *API* do *Google Maps* permite, então, usar todas as funcionalidades disponibilizadas pelo *Google Maps*, como marcar uma posição no mapa, visualizar os mapas na vista de satélite, fazer *zoom* e *pan* no mapa, entre outras coisas, através de programação. Esta *API* possui um manual[2] que contém explicações e exemplos para ajudar os programadores a usá-la.

[2] http://code.google.com/apis/maps/documentation/v3/reference.html

Pode dizer-se que a classe principal é a *Map*. Esta classe é responsável pela construção de um mapa dentro de uma página HTML e por todas as principais opções de controlo do mesmo. Fornece todas as funcionalidades relacionadas com o redimensionamento do mapa, colocar e devolver determinada latitude e longitude, centrar o mapa, fazer *zoom*, fazer *pan*, etc. Permite também, alterar as opções do mapa como mudar a cor de fundo, activar e desactivar as funcionalidades de *zoom* e centrar com duplo clique, activar e desactivar a função de *dragging*, activar e desactivar os atalhos do teclado, entre outras. Existem também um conjunto de classes responsáveis por determinar caminhos entre dois pontos, por obter endereços, etc.

Posto isto, é fácil perceber que já muito trabalho está feito no âmbito dos SIG que pode ser perfeitamente reutilizado para outras aplicações. É também claro que os dados e os requisitos das aplicações geoespaciais estão em constante mutação, portanto os *concerns* espaciais são altamente voláteis, o que faz com que se torne fundamental que haja uma boa gestão destes *concerns* e das suas alterações.

A disponibilização de ferramentas como o *Google Maps* criou uma apetência para a integração da georreferenciação em aplicações *Web* já existentes, onde isso não existia inicialmente. Um exemplo disto poderia ser o Guia da Cidade[3] que poderia ter começado por ser um guia com apenas as moradas dos pontos de interesse ao qual posteriormente seria adicionada informação geográfica.

2.4. Modelação de SIGs

A modelação dos SIG incide sobre a representação dos dados geográficos de forma digital, sendo, de forma genérica, orientada ao tratamento dos dados geográficos (Oliveira, 2009). Adicionalmente, Medeiros *et* al. (1994) consideram que as técnicas de modelação de dados habituais não são adequadas para lidar com a informação geográfica. O facto da maioria dos dados geográficos estarem relacionados com o local onde são válidos, com o tempo de observação e a precisão da observação, traz alguns problemas.

Segundo Medeiros *et* al. (1994), a modelação de dados georreferenciados pelos utilizadores está associada a diferentes percepções do mundo: modelo de camadas (ou *field model*) e modelo de objectos (ou *object model*).

[3] http://www.guiadacidade.pt/portugal/

No modelo de camadas o mundo é uma superfície contínua (*layer*) com características que variam continuamente. Durante o processo de modelação foram criadas entidades que não existem de forma independente, estas são usadas para dividir o campo em áreas, sendo a ênfase deste modelo, o conteúdo destas áreas. O modelo de camadas é processado no formato de tesselação (*tesselation*), isto é, os objectos espaciais são descritos como células numa matriz, e portanto não podem existir espaços para preencher.

No modelo de objectos o mundo é uma superfície plana com objectos reconhecíveis, que existem independentemente de qualquer definição (e.g., um rio). Neste modelo, dois objectos podem estar na mesma posição (e.g., a praia no rio Mira). Na base de dados, as entidades correspondem a estes objectos reconhecíveis. O modelo de objectos é processado em formato vectorial (*vector*), ou seja, os objectos são processados como pontos, linhas e polígonos através de listas de pares de coordenadas. Desta forma, neste modelo, os espaços não têm de estar todos preenchidos, ao contrário do modelo de camadas.

Segundo Borges *et* al. (2001) os primeiros modelos de dados para as aplicações geográficas eram direccionados para as estruturas internas dos SIGs, sendo necessário adaptar os dados espaciais às estruturas oferecidas pelo SIG em causa, não sendo possível representar a realidade da forma mais adequada. Assim, tornou-se evidente que a modelação de dados geográficos necessitava de modelos capazes de compreender melhor o significado dos dados geográficos, oferecendo mecanismos de abstracção mais elevados e independentes da implementação.

Os modelos de dados orientados a objectos, embora muito expressivos (podem representar tanto o modelo em camadas como o modelo de objectos) não contêm primitivas geográficas que possam representar de forma adequada os dados espaciais. Os modelos de dados para aplicações geográficas têm necessidades adicionais relativas aos níveis de abstracção dos dados geográficos utilizados nestas aplicações.

No seu estudo Borges *et* al. (2001) identifica três níveis de abstracção – Figura 2.14 – utilizados nas aplicações geográficas:

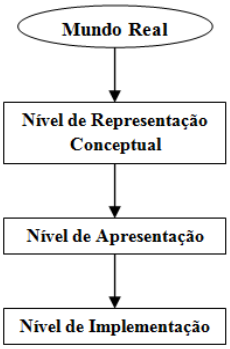

Figura 2.14. Níveis de abstracção das Aplicações Geográficas (Borges *et* al., 2001)

- *Nível Conceptual*. Este nível oferece um conjunto de conceitos formais para modelar as entidades geográficas (como rios, edifícios, estradas e vegetação) com um alto nível de abstracção. É neste nível que são criadas na base de dados as classes básicas que serão associadas às classes de representação espacial.

- *Nível de apresentação.* Este nível oferece as ferramentas necessárias para associar as classes de representação espacial às entidades formais definidas no nível conceptual.

- *Nível de implementação.* Neste nível são definidos os padrões, as formas de armazenamento, as estruturas de dados e as funções *standard* para implementar cada representação, como foi definida no nível conceptual e cada apresentação como foi definida no nível de apresentação.

Na Figura 2.14, além dos três níveis de abstracção descritos anteriormente, está presente um nível inicial do mundo real, que Borges *et* al. (2001) descreve como o nível que contém os objectos geográficos do mundo real, que irão ser representados – os rios, as cidades, a vegetação, entre outros.

Considerando os factores associados à representação da realidade geográfica, Borges *et* al. (2001) obtiveram o seguinte conjunto de requisitos necessários a um modelo de dados para aplicações geográficas.

- Oferecer um elevado nível de abstracção;

- Representar e distinguir os vários tipos de dados presentes nas aplicações geográficas – ponto, linha, área, imagem, etc;

- Representar as relações espaciais, as suas propriedades e as associações;

- Ter a capacidade de especificar regras de integridade espacial;
- Ser independente da implementação;
- Permitir classes georreferenciadas, classes convencionais e ainda os relacionamentos entre elas;
- Adequar-se à sabedoria natural que o ser humano possui relativamente aos dados espaciais, através da representação visual do campo e dos objectos;
- Ser de fácil percepção;
- Recorrer ao conceito de níveis de informação, permitindo que uma entidade geográfica seja associada a vários níveis de informação;
- Representar as várias visualizações possíveis de uma entidade geográfica;
- Ter a capacidade de exprimir o desenrolar do tempo.

Para modelar os SIG têm sido usadas técnicas orientadas a objectos (Gordillo *et* al., 1997) e a programação orientada a aspectos (Zipf *et* al., 2003). No entanto, dado que os requisitos dos SIG, em especial na *Web*, apresentam uma grande propensão para a mudança, é fundamental que se tenha um método de modelação capaz de garantir a fácil gestão e reutilização do sistema.

2.5. Sumário

Ao longo deste capítulo explicou-se o que são os sistemas de informação geográfica, a evolução dos SIG para *Web*-SIG, demonstrando-se a importância destes sistemas. São apresentados, também os diferentes tipos de *Web*-SIG, bem como as vantagens e desvantagens de cada um deles. Em seguida foram introduzidos os dois principais tipos de Arquitectura dos Sistemas *Web*-SIG. Finalmente, foi introduzido o tema das *Maps API*, onde se usa o exemplo do *Google Maps* para mostrar a importância que as *Maps API* têm tido no desenvolvimento e surgimento de novas aplicações *Web*-SIG. Este exemplo serve também para demonstrar como é possível e vantajoso identificar chamadas a funções destas *API* que podem constituir padrões de reutilização.

Seguidamente, abordou-se o tema da modelação de aplicações SIG em que se discutem os modelos de dados para aplicações geográficas, bem como os níveis de abstracção dos

modelos de dados para estas e, finalmente, apresenta-se um conjunto de requisitos necessários a um modelo de dados para aplicações geográficas.

Posto isto, revela-se a necessidade de modelar as aplicações geográficas com requisitos, incidindo nos requisitos voláteis destas aplicações de forma a poder controlar as alterações que os requisitos destas aplicações possam sofrer.

3. Engenharia de Requisitos e Reutilização

A Engenharia de Requisitos (Sommerville, 2007) é todo o processo de elicitação, análise, documentação e validação de requisitos do sistema. Este processo ajuda a que haja uma melhor compreensão do problema a ser resolvido por parte dos engenheiros de *software* (Pressman, 2005).

A Engenharia de Requisitos (Pressman, 2005) começa por ter uma fase de iniciação onde se define o âmbito e a natureza do problema a ser resolvido. Segue-se a fase de elicitação onde o cliente e a equipa de engenheiros de requisitos definem o que é necessário para resolver o problema. Depois vem a fase de elaboração, onde se especificam e modificam os requisitos básicos. Assim que o cliente consegue definir o problema inicia-se a fase de negociação onde se definem quais as prioridades e o que é fundamental. Nesta altura, o problema é finalmente especificado e validado, de forma a assegurar que tanto o cliente como os engenheiros de requisitos têm a mesma compreensão do problema.

3.1. Requisitos

Um requisito é uma propriedade ou comportamento que um produto ou serviço deve possuir. O conceito de requisito refere-se à definição de uma característica que um sistema ou módulo de um sistema deve satisfazer para obter o resultado pretendido (Sommerville *et* al., 1997).

Ao longo dos tempos surgiram diversas formas de classificar os requisitos. Por exemplo, Sommerville (2007) propôs dividi-los em dois tipos principais:

1. **Requisitos do Utilizador** – Estes são diagramas e descrições em linguagem natural dos serviços fornecidos pelo sistema e as suas restrições operacionais. Destinam-se aos Clientes;

2. **Requisitos do Sistema** – Estes constituem um documento estruturado com a descrição detalhada das funções, serviços e restrições operacionais do sistema. Definem o que deve ser implementado, podendo fazer parte de um contracto entre o Cliente e a empresa que implementa.

Sendo estes últimos classificados como:

a. **Requisitos funcionais** – são características dos serviços que o sistema deve fornecer, como o sistema deve reagir a determinados *inputs* e como se deve comportar em determinadas situações;

b. **Requisitos não-funcionais** – são restrições aos serviços ou funções disponibilizadas pelo sistema;

c. **Requisitos de domínio** – reflectem características do domínio e podem ser funcionais ou não-funcionais.

No entanto, entre tantas classificações que foram surgindo notou-se um consenso no que respeita à existência de duas classes de requisitos, os **funcionais** e os **não-funcionais**.

Os requisitos funcionais são descrições das funcionalidades que o sistema deve oferecer, de uma forma completa e consistente. Descrevem, também como o sistema deve reagir e como se deve comportar em determinadas situações.

Os requisitos não-funcionais são atributos de qualidade e restrições sobre as quais o sistema deve funcionar. Têm um papel crítico no desenvolvimento do software, pois são difíceis de testar, sendo avaliados de forma subjectiva. Temos como exemplos de requisitos não-funcionais a Usabilidade, Segurança, Tempo de Resposta e Desempenho.

Existem ainda os requisitos transversais (*crosscutting concerns*) e os voláteis que serão enfatizados nesta dissertação, descritos a seguir.

3.1.1. Requisitos transversais

O Desenvolvimento de *Software* Orientado a Aspectos (DSOA ou AOSD - *Aspect-oriented Software Development*) é uma abordagem para desenvolvimento de *software* em que as técnicas orientadas a aspectos são integradas com as técnicas de desenvolvimento tradicionais, fornecendo novas abstracções e mecanismos para suportar a modularização dos *crosscuting concerns*. (Henderson-Sellers, 2007 e Sutton *et* al., 2005).

Esta abordagem de desenvolvimento de *software* surgiu para fazer face ao problema da reutilização de componentes. Assim, o seu grande objectivo é endereçar os *crosscutting concerns* fornecendo meios para a sua identificação, separação, representação e

modularização, com vista a tornar os programas mais fáceis de manter e de reutilizar (Sommerville, 2007 e Rashid *et* al., 2003).

Esta separação e modularização dos *crosscutting concerns* é conseguida através da introdução do conceito de aspectos, onde os *crosscutting concerns* são separados e encapsulados como aspectos para serem compostos com outros módulos de forma automática (Fu *et* al., 2009).

Posto isto, percebe-se que o ponto central do DSOA são os *concerns*. Segundo Sutton *et* al. (2005), um *concern* é qualquer assunto de interesse num sistema de *software*. Estes são fundamentalmente conceptuais. Um *concern* diz-se ser *crosscutting* (transversal) se estiver *tangled* (emaranhado) com outro *concern* num único módulo ou se está *scattered* (disperso) por diversos módulos do sistema (Figueiredo *et* al., 2005).

Os *crosscutting concerns* (Rashid *et* al., 2003) estão encapsulados em módulos, conhecidos como Aspectos. Isto resulta num suporte mais robusto para a modularização, reduzindo desta forma o custo do desenvolvimento, da manutenção e da evolução.

Segundo Sommerville (2007), os aspectos encapsulam funcionalidades que atravessam (*crosscuts*) e coexistem com outra funcionalidade incluída no sistema. A grande vantagem das abordagens orientadas a aspectos é a capacidade de separação dos *concerns*. A separação dos *concerns* em elemento independentes, em vez de incluir diferentes *concerns* na mesma abstracção lógica é uma boa prática de Engenharia de *Software*. Representando os *crosscutting concerns* como aspectos, estes podem ser compreendidos, reutilizados e modificados de forma independente.

As vantagens do DSOA são: uma maior separação dos *concerns*, minimizando a replicação de código, melhorando a coesão entre os diversos módulos, a redução do entrelaço do sistema, e consequente aumento da possibilidade de reutilização e facilidade de evolução no desenvolvimento de sistemas de *software* complexos (Figueiredo *et* al., 2005).

Neste trabalho vão usar-se os mecanismos de DSOA para modelar *concerns* espaciais voláteis em aplicações SIG.

3.1.2. Requisitos Estáveis e Requisitos Voláteis

Actualmente, depois de todo este processo, é inevitável que os requisitos de um sistema se alterem. Isto acontece devido à constante mudança de toda a envolvência empresarial onde os

sistemas se inserem. Assim, para fazer face a estas alterações é necessário que exista uma boa gestão dos requisitos.

Os requisitos estáveis ou permanentes são aqueles que estão relacionados com a essência do sistema e do domínio da aplicação, são portanto, requisitos razoavelmente estáveis que derivam do processo de negócio da organização. Alteram-se mais lentamente que os requisitos voláteis.

Os requisitos voláteis são mais propícios a sofrer alterações do que outros. Estes requisitos são específicos da instanciação do sistema num ambiente em particular e para um determinado cliente. Dada a sua propensão para a alteração devem ser altamente independentes dos restantes requisitos.

Os requisitos voláteis podem ser de diversos tipos (Kotonya *et* al., 2004):

- Mutáveis: são requisitos que se modificam devido a alterações no ambiente em que o sistema opera;
- Emergentes: são requisitos que não podem ser completamente definidos na fase de especificação do sistema, pois alteram-se durante o desenvolvimento, isto é, vão surgindo à medida que o conhecimento do cliente aumenta no desenvolvimento do sistema;
- Consequentes: são requisitos que se baseiam em pressupostos sobre o uso do sistema. Quando este começa a ser usado alguns pressupostos mostram-se errados. Os utilizadores ao usarem vão descobrir novas formas de usar as funcionalidades do sistema. Tal situação origina novas necessidades dos utilizadores devido a alterações no sistema;
- Compatíveis: são requisitos que dependem de outros sistemas ou processos, portanto, se algum desses sistemas ou processo evoluir estes requisitos evoluem também.

Para uma boa Gestão de Requisitos é importante classificá-los para identificar os mais voláteis, de forma a antecipar possíveis alterações nos requisitos. Esta identificação permite, também que os programadores do sistema implementem estes requisitos em componentes independentes, de modo a que as possíveis futuras alterações tenham um impacto mais reduzido no sistema.

3.2. Reutilização de Requisitos

Nesta dissertação as técnicas de reutilização de requisitos a ser tratadas são: um mecanismo de modelação com aspectos utilizando a notação MATA; a análise de domínio e o modelo de *features*; e a utilização de padrões de análise.

3.2.1. MATA – Modeling Aspects using a Transformation Approach

MATA consiste (Whittle *et* al., 2007) numa abordagem de modelação orientada a aspectos que possui mecanismos de composição baseados em transformação de grafos que usam modelos UML, como Diagramas de Classes e de Sequência. Estes modelos traduzem cenários que são uma técnica usada para especificar o comportamento de um sistema.

Em MATA começa-se por identificar um modelo base e um modelo aspectual, sendo que a base é intersectada pelo aspectual. Compondo estes dois modelos e usando as regras de composição da abordagem obtém-se o modelo composto. Estas regras de composição são definidas no modelo aspectual, de forma a indicar onde serão inseridos os novos elementos no modelo base.

Para definir as regras de composição foram criados 3 estereótipos:

- «create» → indica a criação de um elemento na base;
- «delete» → indica a remoção de um elemento da base;
- «context» → indica que determinado elemento não será influenciado pelos estereótipos «create» e «delete», servindo para fazer o *pattern matching* com elementos da base.

Para a obtenção do modelo composto é feito um *pattern matching* entre os modelos base e aspectual. Para isto, estabelecem-se as correspondências entre os elementos de cada modelo. As variáveis MATA usadas para instanciar os elementos dos modelos aspectuais com elementos concretos da base, como parte inicial da composição, são definidas usando o prefixo "|".

Abaixo apresenta-se um exemplo, com cenários representados por digramas de sequência, para clarificar o processo desta abordagem. Na Figura 3.1, temos o cenário base, que consiste num diagrama de sequência UML. Na Figura 3.2, temos o cenário aspectual, que como se vê usa os estereótipos do MATA, indica-se portanto que "p" não será afectado por nenhum «create» ou «delete», e "|m" é uma variável MATA que também não será afectada por nenhum «create» ou «delete». Indica-se também que a entidade "c:C" será criada quando for

feita a composição com a base, bem como o fragmento ALT. Na Figura 3.3, temos o Cenário Composto que segue as regras de composição definidas no Cenário Aspectual.

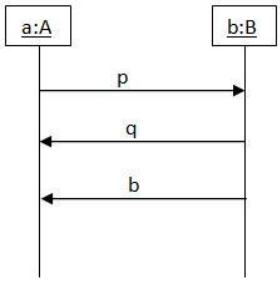

Figura 3.1. Cenário base

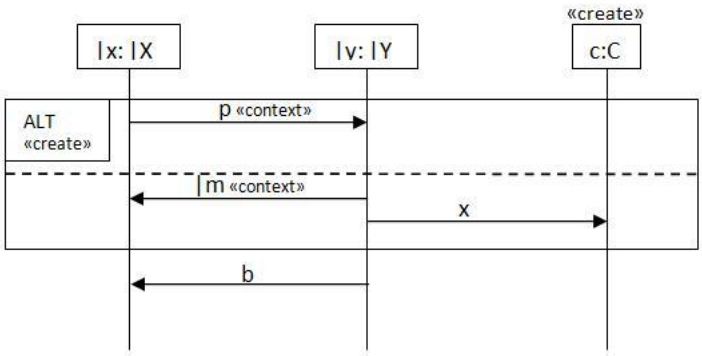

Figura 3.2. Cenário aspectual

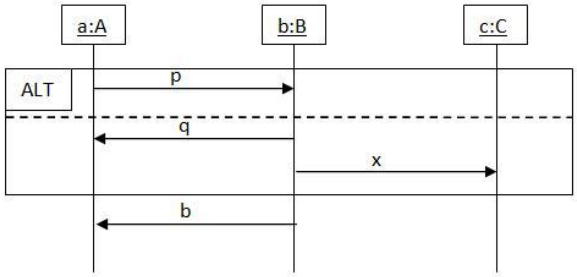

Figura 3.3. Cenário Composto

Esta abordagem tem a particularidade de não usar *joinpoints* – pontos bem definidos onde se adiciona código ou comportamento do aspecto – explícitos. Assim, cada modelo pode ser um *joinpoint* e a sua composição é um caso especial de transformação de modelos. A principal vantagem desta abordagem é conseguir lidar com a identificação prévia dos erros, inconsistências ou ambiguidades.

3.2.2. Análise de Domínio

Neste trabalho, o domínio de estudo será o domínio dos Sistemas de Informação Geográfica (SIG), mais concretamente os sistemas que envolvem a visualização e manipulação de informação georreferenciada na *Web*. Pretende-se analisar este domínio para identificar diversas características que permitam a reutilização de soluções usadas pela comunidade SIG. Para esse fim, vai começar-se por usar uma técnica de análise de domínio orientada a *features*, o *Feature Model*. E depois vão ser usadas as técnicas de descrição de padrões que podem ser aplicadas em SIG.

De seguida descrevem-se os principais conceitos de análise de domínio e diagrama de *features*. Sendo depois, proposto um modelo de *features* para SIG, que vai ser útil para nos guiar na definição dos artefactos a reutilizar para este domínio de aplicações. Iremos depois discutir os principais conceitos e características da análise de padrões. Finalmente, são aplicados padrões a SIG.

3.2.2.1. Conceitos de Análise de Domínio

O conceito de *software* reutilizável existe desde o início da programação de computadores, no entanto, só se tornou objecto de investigação na Engenharia de *Software* por volta dos anos 80 (Arango, 1989).

A área de "Análise de Domínio" foi introduzida por Neighbors (1961) para se referir às actividades de identificação, análise e especificação dos requisitos comuns, pertencentes a uma aplicação de domínio específico, para que sejam reutilizados na especificação e construção de outros projectos de *software* que venham a ser criados (Arango, 1989; Prieto-Díaz, 1990; Pressman, 2005).

Neighbors (Prieto-Díaz, 1990) introduziu também, o conceito de "Analista de Domínio", como a pessoa responsável por identificar e definir os padrões de análise reutilizáveis, as classes de análise, e a informação relacionada que pode vir a ser usada por outras pessoas que trabalhem em aplicações semelhantes (Pressman, 2005).

O analista de domínio tem de ter um elevado conhecimento sobre todas as áreas de interesse para o processo, Neighbors considera mesmo que este deve ser um *Domain Expert*. Além disso, tem de ter a capacidade de comunicar com todas as outras pessoas envolvidas no processo (Prieto-Díaz, 1990).

O processo da Análise de Domínio tem como objectivo disponibilizar toda a informação relacionada com o desenvolvimento, para que quando o engenheiro de *Software* tiver de decidir se reutiliza, ou não, determinado componente, tenha acesso à informação que lhe permita compreender todo o contexto que levou o *designer* que criou o componente a criá-lo de determinada forma. Isto permite tomar uma decisão consciente e informada e tornar a reutilização mais eficaz (Prieto-Díaz, 1990).

Na figura 3.4 (Pressman, 2005), podemos ver os *inputs* e *outputs* envolvidos no processo de análise de domínio. As fontes de conhecimento do domínio são analisadas, de forma a identificar objectos que possam ser reutilizados em todo o domínio.

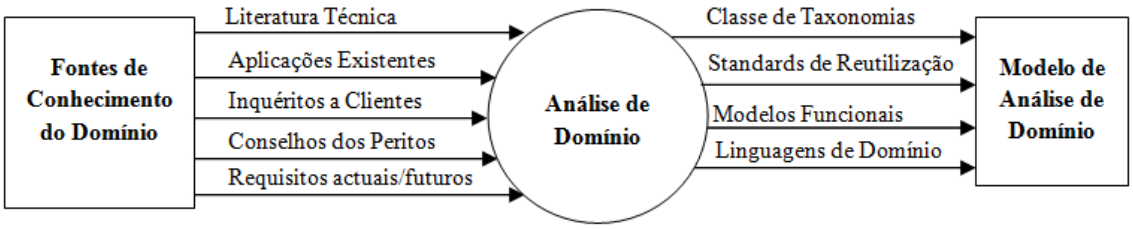

Figura 3.4. *Inputs* e *Outputs* da Análise de Domínio (Pressman, 2005)

Arango (1989) identifica duas etapas importantes na Análise de Domínio: a Análise Conceptual; e a Análise Construtiva. Ambas consistem na identificação e obtenção de informação. No entanto, no que diz respeito à Análise Conceptual, a informação obtida destina-se à especificação dos sistemas do domínio. No caso da Análise Construtiva, a informação servirá para a implementação das especificações obtidas na Análise Conceptual.

Prieto-Díaz (1990) apresenta no seu artigo um processo básico para conduzir a análise de domínio. Este processo foi testado, com sucesso, inúmeras vezes em projectos da *IBM Federal Systems*. O processo consiste em três passos que vão repetir-se várias vezes para diferentes tipos de componentes:

1. Identificação de entidades reutilizáveis;
2. Abstracção ou generalização;
3. Classificação e Catalogação para outras reutilizações.

As informações obtidas a partir do modelo de domínio podem ser de grande relevância, pois permitem ao reutilizador atingir níveis específicos de desempenho nas tarefas de construção de *software* (Arango, 1989).

3.2.2.2. Análise de Domínio Orientada a *Features* - Diagrama de *Features*

De acordo com Mei *et* al. (2003), a modelação dos requisitos de domínio tem como objectivo obter um conjunto de requisitos reutilizáveis, o que implica a utilização de uma *Framework* de modelos de requisitos poderosa o suficiente para suportar a reutilização. Para alcançar estes objectivos foi sugerido o recurso a abordagens orientadas para *features*.

Segundo Kang *et* al. (1990), da análise de domínio resultam modelos usados para desenvolver aplicações no domínio em causa.

O conceito de Modelo de *Features* – na Figura 3.5 é apresentado o exemplo de um modelo de *Features* – foi introduzido na Engenharia de Domínio pela Análise de Domínio Orientada a *Features*, e desde então tem sido usado como modelo principal, em conjunto com outros modelos como o de Use Cases e o de Objectos, para construir o modelo de requisitos do domínio. (Mei *et* al., 2003)

A análise do modelo de *features* que Kang *et* al. (1990) apresentam no seu artigo é bastante clara e foca as principais características do mesmo. Vamos assim debruçar-nos sobre esse mesmo artigo.

Kang apurou que quando se produz um diagrama de *features*, produz-se um diagrama estrutural que mostra a decomposição hierárquica das *features*, com a indicação de se são opcionais ou alternativas, a sua definição, num dicionário de terminologia do domínio, onde deverá constar o significado de todos os termos necessários para a compreensão das *features* incluídas no modelo, e as regras de composição.

O modelo de *features* é um bom meio de comunicação, porque ao apresentar-se de forma visual faz com que seja mais fácil para os utilizadores compreendê-lo.

O modelo de *features* serve como meio de comunicação entre os utilizadores e os programadores, e para Kang *et* al. (1990), para cada uma das partes tem um significado diferente:

- Para os utilizadores mostra o que são as *features standard*, que outras *features* podem escolher, e quando as podem escolher.
- Para os programadores, indica o que precisa de ser parametrizado nos outros modelos e a arquitectura do *software*, e como a parametrização deve ser feita.

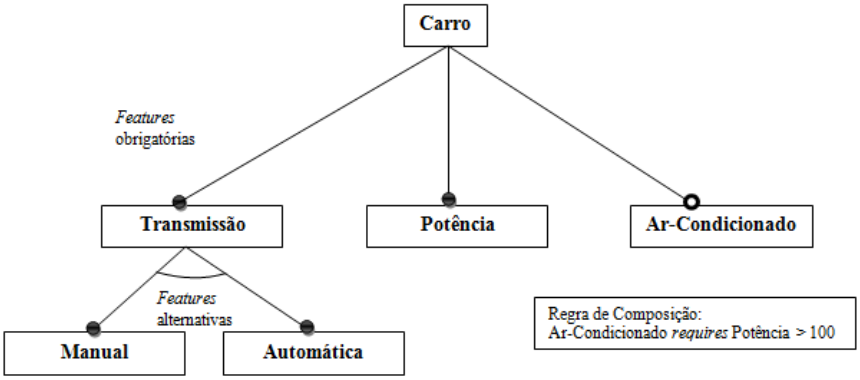

Figura 3.5. Exemplo de Modelo de *Features* (Kang *et* al., 1990)

Kang definiu, os principais conceitos do modelo de *features* da seguinte forma:

- *Features*: são atributos do sistema que afectam directamente os utilizadores finais.
- Modelo de *features* (Figura 3.5): representa as *features standard* de uma família de sistemas no domínio e as relações entre elas. Cada *feature* do modelo deve ter um nome diferente dos das restantes *features* e a definição de cada uma delas deve ser incluída no dicionário de terminologia do domínio.
- *consists of*: É uma relação estrutural que representa um grupo lógico de *features*.
- *Features* opcionais: indicam *features* que são opcionais, como podemos ver na Figura 3.5, em que temos o Ar-Condicionado como uma *feature* opcional, pois um carro pode ter ou não um aparelho de ar-condicionado.
- *Features* alternativas: indicam *features* em que apenas uma sub-*feature* pode ser seleccionada. São consideradas especializações de uma categoria mais geral. O termo *alternative feature* é usado para indicar que não pode ser feita mais que uma especialização para o sistema. No entanto, os atributos de uma *feature* geral são herdados por todas as suas especializações. Na Figura 3.5, temos o exemplo de *features* alternativas, o caso do tipo de transmissão que só pode ser ou manual ou automático.

As *features* opcionais e alternativas de um grupo devem estar indicadas graficamente no modelo de *features*.

As regras de composição definem a semântica existente entre as *features* que não estão expressas no diagrama de *features*. Kang *et* al. (1990) explicam-nas da seguinte forma:

- *mutually exclusive with*: indica que as *features* opcionais e alternativas, em causa, não podem ser seleccionadas quando determinada *feature* for escolhida.

- *Requires*: indica que as *features* opcionais e alternativas, em causa, devem ser seleccionadas quando determinada *feature* for escolhida. No diagrama de *features* da Figura 3.5 é apresentado um exemplo de uma regra de composição do tipo *requires*, em que a *feature* Ar-Condicionado apenas pode ser seleccionada se a *feature* Potência for superior a 100.

3.2.2.3. Diagrama de *Features* para uma aplicação SIG na *web*

Na Figura 3.6 é apresentado um diagrama de *Features* obtido para adicionar determinadas características geoespaciais a alguns objectos de uma aplicação *Aspect-Web*. Como podemos ver, uma aplicação *Aspect-Web* poderá possuir todas as características apresentadas no diagrama da Figura 3.6, mas não existe nenhuma que tenha obrigatoriamente de as ter. No entanto, por exemplo, se a aplicação tiver a capacidade de adicionar nova informação descritiva a dados espaciais, terá de ter obrigatoriamente um tipo de media associado, neste caso temos o exemplo de vídeos, uma interface e estruturas de dados, podendo ter uma estrutura de *links* que teria de ter obrigatoriamente *links* para vídeos. O diagrama da Figura 3.6 mostra que uma aplicação *Aspect-Web* pode ter todas as funcionalidades nele apresentadas.

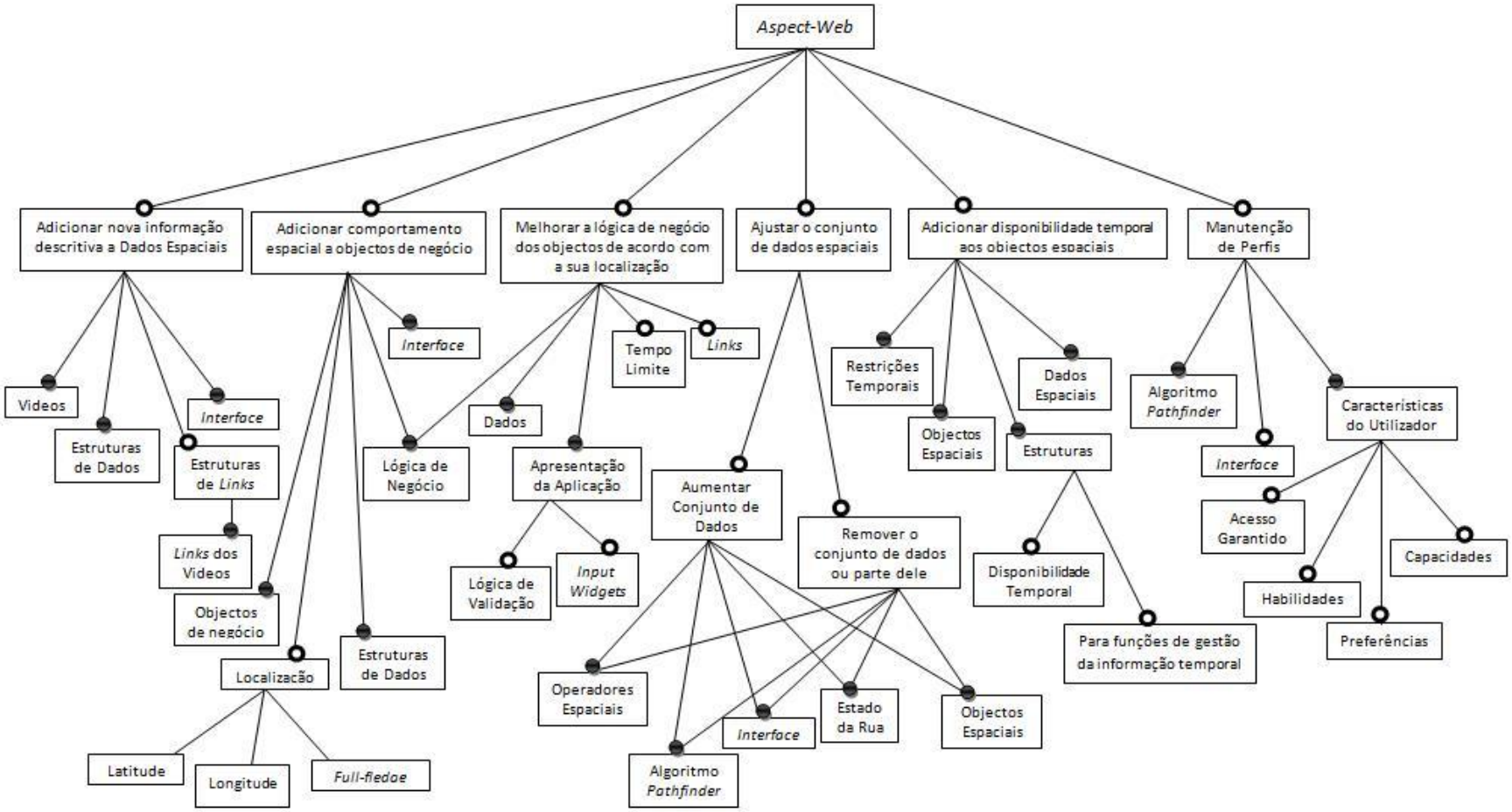

Figura 3.6. *Feature Model* para a Aplicação *Aspect-Web*

No fragmento do diagrama da Figura 3.6 – apresentado na Figura 3.7 – temos a característica de adicionar comportamento espacial a objectos de negócio. Suponhamos que tínhamos uma empresa de autocarros que pretendia mostrar aos seus utilizadores a localização dos seus autocarros. Para isto, precisaríamos de ter: objectos de negócio, os autocarros; a localização destes objectos; uma estrutura de dados, isto é, uma base de dados de mapas, com os percursos e as paragens dos autocarros; a lógica de negócio; e finalmente a interface para mostrar a localização dos autocarros.

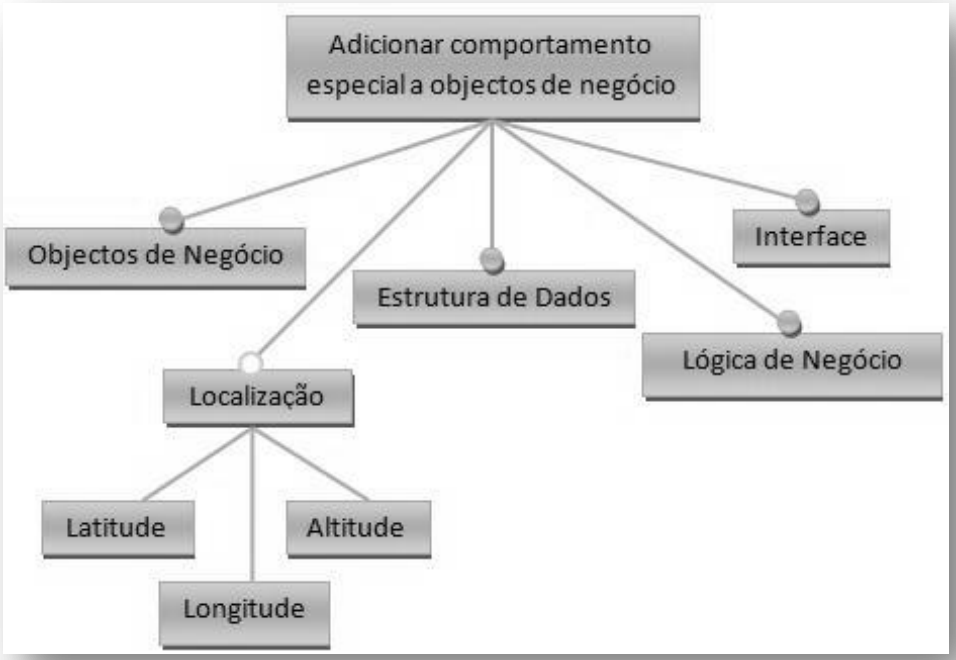

Figura 3.7. Fragmento do *Feature Model* para a Aplicação *Aspect-Web*

3.2.3. Padrões de Análise

De forma a complementar a análise de domínio apresentada na secção anterior, e para permitir a reutilização de modelos, vão usar-se padrões de análise.

Na última década, os padrões têm vindo a ser, cada vez mais, vistos como uma técnica para aumentar a qualidade do desenvolvimento de *software* e diminuir os custos e o tempo do desenvolvimento. Um padrão, de forma genérica, é descrito como algo que pode ser usado como modelo em diversos contextos (Hamza *et* al., 2004).

Coad *et* al. (1995) definiram padrões como templates de objectos que podem ser usados inúmeras vezes por analogia. Assim, um padrão obtido de determinado projecto pode ser

generalizado de forma a poder ser aplicado para modelar o mesmo problema em outras aplicações e domínios (Hamza *et* al., 2004).

Os padrões mais utilizados são os de desenho e, em níveis mais abstractos, de análise e de requisitos.

Segundo Harrer *et* al. (2002), os padrões de desenho são estruturas de desenho comuns que são úteis na criação de *software* reutilizável. À semelhança deste, Konrad *et* al. (2002) dizem que os padrões de desenho propõem esqueletos de soluções para problemas de desenho habituais. Sendo o esqueleto da solução descrito de tal forma que o desenho pode ser usado por outros projectos, sendo que para cada aplicação terão de ser criadas as devidas restrições específicas do projecto. De acordo com Konrad *et* al. (2004), os padrões de desenho estão próximos da implementação, e geralmente, preocupam-se com tarefas, objectos activos, *scheduling*, a distribuição do tempo que leva a correr os componentes, endereçamento de propriedades não funcionais e o detalhe da definição das *interfaces*.

Os padrões de análise servem para construir modelos de análise. As principais preocupações destes padrões são os modelos conceptuais, a flexibilidade e a capacidade de reutilização dos sistemas que resultam da aplicação destes padrões.

Os padrões de requisitos têm como principais objectivos a documentação das necessidades dos utilizadores e a especificação do comportamento geral do sistema (Pantoquilho *et* al., 2003). Estes padrões podem também, ser usados para encontrar requisitos funcionais e não-funcionais pertencentes a um determinado sistema, podendo posteriormente ser especificados com mais detalhes ao nível do *design* e da implementação, podendo até recorrer-se, para tal, a padrões de *design* e de arquitectura (Konrad *et* al., 2002).

Geyer-Schulz e Hahsler identificaram duas vantagens no uso de padrões de análise (Pressman, 2005):

1. Aceleram o desenvolvimento de modelos de análise abstractos, que captam os principais requisitos do problema em concreto, fornecendo modelos de análise reutilizáveis.

2. Facilitam a transformação de modelos de análise em modelos de *design*, sugerindo padrões de *design* e soluções fiáveis para problemas comuns.

Assim, os padrões permitem que se beneficie da experiência de alguém que esteve em semelhante situação, o que é especialmente importante nos dias que correm, pois cada vez

mais os sistemas de *software* tendem a crescer exponencialmente, sendo também cada vez maior a necessidade de os manter extensíveis e fácies de gerir (Harrer *et* al., 2002).

3.3. Sumário

Ao longo deste capítulo foram abordados alguns temas relativos à Engenharia de Requisitos. Nomeadamente, foram apresentados os principais conceitos nesta área, que nos permitissem ter uma base teórica para avançar com o objectivo desta dissertação. Desta forma, foi feita uma breve explicação sobre os diversos tipos de requisitos existentes, dando-se alguma ênfase aos requisitos transversais e aos requisitos voláteis que vão ter um papel de relevo nesta dissertação. Foi apresentada uma técnica de modelação orientada a Aspectos, tendo sido destacada a sua grande capacidade para separar os *concerns* transversais de uma aplicação, pois terá interesse para a solução do problema proposto. Foram também apresentadas as diversas fases que constituem a Engenharia de Requisitos.

Depois de assentes todos estes conceitos e processos, torna-se evidente que, para conseguir obter um método de modelação dos *concerns* voláteis presentes nas aplicações SIG, é fundamental que se tenha um conhecimento profundo do domínio em causa. Assim, ao longo deste capítulo explicámos também no que consiste a análise de domínio, bem como a sua importância. Referimos ainda técnicas para a fazer, nomeadamente, o Modelo de *Features*. Depois, desenvolvemos uma breve explicação relativamente ao uso de padrões para permitir a reutilização. Feito isto, resta-nos explicitar como estas técnicas são fundamentais para um desenvolvimento mais simples e fácil de manter, bem como de reutilizar. Estas técnicas permitem que os programadores partilhem as suas experiências, evitando que um programador tenha de refazer todo o trabalho já desenvolvido por um colega seu.

4. Trabalho relacionado

No domínio da modelação de aplicações SIG na *Web* não existem muitos trabalhos desenvolvidos. No entanto, existem alguns trabalhos desenvolvidos em temas relacionados com o que se vai desenvolver nesta dissertação:

- Desenvolvimento de uma abordagem baseada em Aspectos para modelar aplicações SIG – *Modelação de Aplicações SIG com Aspectos* (Oliveira, 2009);
- Proposta da integração das ontologias do domínio SIG com a orientação a aspectos – *Is aspect-orientation a new paradigm for GIS development? On the Relationship of Geoobjects, Aspects, and Ontologies* (Zipf *et* al., 2003).
- Aplicação de padrões de análise no domínio SIG – *Applying Analysis Patterns in the GIS Domain* (Lisboa *et* al., 1998);
- Apresentação de uma abordagem para desenhar aplicações SIG, usando padrões de desenho – *Developing GIS Applications with Objects: A Design Patterns Approach* (Gordillo *et* al., 1999).
- OMT-G (*Object Modelling Techniques for Geographic Applications*) que consiste num modelo de dados para aplicações geográficas – OMT-G: *An Object-Oriented Data Model for Geographic Applications* (Borges *et* al., 2001).

Todos estes trabalhos tiveram um propósito que servirá para corroborar o desta dissertação. O que se pretende é que sejam identificados alguns requisitos voláteis reutilizáveis nas aplicações SIG, para que depois sejam descritos com um *template* de padrão – nesta dissertação foram descritos dois padrões – e, finalmente, juntando todos os padrões obtidos poderá construir-se um catálogo de padrões, sendo que para se proceder à modelação, se irá recorrer a mecanismos de DSOA.

Uma característica que marca as aplicações SIG na *Web* é a sua volatilidade, ao nível dos requisitos. No entanto, em qualquer aplicação, mesmo em SIG, existe um conjunto de *concerns* que se mantêm. Assim, surge a necessidade de isolar estes *concerns* (espaciais) para que, quando uma aplicação evoluir, apenas haja impacto em determinados *concerns* e não em toda a aplicação. É ainda necessário que os *concerns* voláteis destas aplicações sejam modelados de forma a poderem ser inseridos, alterados ou removidos da aplicação.

4.1. Abordagem Orientada a Aspectos para Modelação de Aplicações SIG

Oliveira (2009) desenvolveu uma abordagem orientada a Aspectos para modelar aplicações SIG, isto é, foi desenvolvido um modelo para identificar, modularizar e compor os *crosscutting concerns*, mais precisamente os *concerns* espaciais de uma aplicação, com o propósito de fazer face à modelação de baixo nível usada nas aplicações SIG, que pode gerar componentes espalhados, criando dificuldades ao nível da modularidade e da capacidade de reutilização do sistema.

Numa primeira fase, como apoio à dissertação, foi usada a aplicação *MetaCarta*, isto é, foi feita uma análise inicial a esta aplicação para identificar alguns aspectos importantes das aplicações SIG na *Web*. A aplicação *MetaCarta* é uma aplicação SIG *Web* que permite manipular mapas, pesquisar notícias utilizando um mapa, aplicar restrições à pesquisa, como um determinado local ou uma determinada data, especificar a categoria da notícia e visualizar as notícias relacionadas por um assunto ou local de ocorrência. Depois de identificados os aspectos fundamentais destas aplicações foi elaborado o modelo que se pretendia, como podemos ver na Figura 4.1.

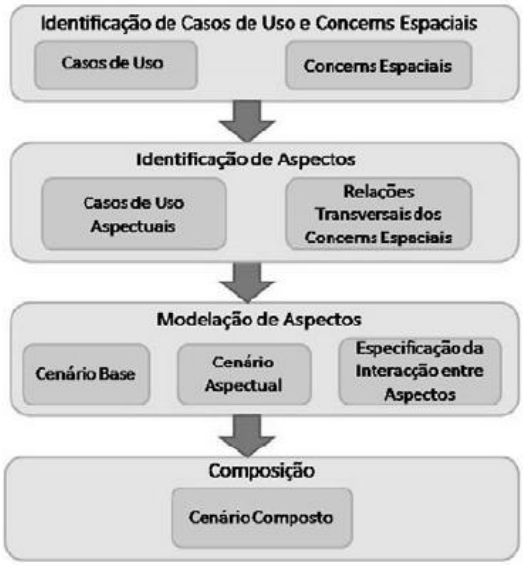

Figura 4.1. Abordagem para modelar aplicações SIG com aspectos (Oliveira, 2009)

Este modelo é constituído por 4 etapas:

1. Identificação dos Casos de Uso e *Concerns* Espaciais;
2. Identificação de Aspectos;
3. Modelação de Aspectos;
4. Composição.

Criado o modelo, seguiu-se a etapa de validação do mesmo. Para tal, foi usada a aplicação SIG *Maps@web*. Esta aplicação tem como objectivo localizar espacialmente serviços em áreas urbanas através de uma estrutura genérica. A validação do trabalho foi constituída pela modelação desta aplicação, seguindo as etapas do modelo sugerido. Para avaliar o desempenho do modelo em situação de alterações na aplicação, testou-se também a possibilidade de adicionar um novo requisito, sendo repetida toda a modelação de acordo com o modelo.

Assim, esta dissertação permitiu obter uma abordagem orientada a aspectos genéricos, que permite modelar aplicações SIG *online*. Uma das conclusões do trabalho de Oliveira (2009) foi que a aplicação deste modelo oferece vantagens como a modularidade, a eficiência, a extensibilidade e a reutilização das aplicações. Nesta dissertação pretende-se fazer o mesmo, mas os *concerns* a identificar e a modelar serão os *concerns* espaciais voláteis que sejam reutilizáveis. A identificação dos requisitos é feita mediante um conhecimento profundo do domínio SIG, obtido através de técnicas de análise de domínio.

4.2 Integração das ontologias do Domínio SIG com a Orientação a Aspectos

De acordo com Zipf *et* al. (2003), o conhecimento do domínio consiste em conceitos e relações entre os conceitos, bem como as restrições sobre os conceitos e as relações, e regras que determinam como inferir ou calcular novos conceitos e relações. Existe uma analogia entre ontologias – definição do conhecimento do domínio em termos dos conceitos, regras e restrições – e a programação orientada a aspectos.

Coordenar as ontologias com a orientação a aspectos no domínio SIG, pode trazer muitas vantagens, como uma maior facilidade de gestão do desenvolvimento dos SIG (Zipf *et* al., 2003).

Os autores defendem que aplicando a programação orientada a aspectos para modelar o domínio como um programa aspectual e o algoritmo como o programa base, permite que

ambos evoluam independentemente um do outro. O que permite resolver alguns problemas de manutenção e reutilização dos SIG.

Neste artigo, os autores apresentam-nos um exemplo para ajudar a compreender as vantagens existentes na orientação a aspectos para o domínio SIG. Este exemplo combina a maioria dos construtores da linguagem AspectJ e que ilustra as suas capacidades para lidar com *crosscutting concerns*.

A utilização de aspectos para acrescentar os modelos de dados SIG e as bibliotecas de classes tem várias vantagens como:

- Extensão da funcionalidade de uma biblioteca SIG existente sem modificar as suas fontes;
- Combinar uma biblioteca SIG já existente com uma biblioteca de outro domínio;
- Alteração do comportamento das classes dispersas pelo *source code* num único aspecto.

Os benefícios de considerar o conhecimento do domínio como um aspecto são a separação do conhecimento do domínio da sua aplicação, tornando o processo de programação menos complexo, libertando os programadores de interligar os aspectos do domínio com os componentes do programa de forma manual. Isto leva a que as aplicações sejam mais fáceis de compreender e de manter, uma vez que são menos desordenadas relativamente ao código específico do domínio.

Assim, defende-se que aplicando a programação orientada a aspectos para modelar o domínio como um programa aspectual e o algoritmo como o programa base, permite que ambos evoluam independentemente um do outro. O que permite resolver alguns problemas de manutenção e reutilização dos SIG. Assim, revela-se a importância de, no trabalho desta dissertação, fazer uma boa análise do domínio SIG para que se consiga identificar os padrões voláteis de reutilização.

4.3. Padrões de Análise no Domínio SIG

Segundo Lisboa *et* al. (1998), o domínio dos SIG pode beneficiar da abordagem de Padrões, pois muitos conjuntos de entidades geográficas são recorrentes em diversas aplicações SIG. É considerado que a abordagem de padrões oferece uma contribuição importante para auxiliar o analista na fase inicial do projecto. Com base nela, é possível desenvolver mecanismos de *software* capazes de integrar definições de padrões tanto em modelos de informação

geográfica existentes como em novos e também capazes de traduzir estas definições em linguagens de definição aceites pelos produtos SIG comerciais.

A utilização de padrões de análise implica algumas alterações na forma como a modelação de dados conceptuais é feita actualmente. Além da habitual análise de requisitos, o *designer* deve pesquisar em uma ou mais bibliotecas de padrões, tentando encontrar soluções já existentes para a modelação dos requisitos que identificou através da análise de requisitos.

Na verdade, o estudo de padrões existentes para alguns domínios de aplicações pode dar ao *designer* uma perspectiva útil sobre a realidade geográfica que está a ser modelada. Desta forma, o processo de análise de requisitos pode ser realizado com muito mais confiança, bem como compreensão.

Além de permitir a reutilização de conhecimento e experiência, os padrões podem melhorar a qualidade da documentação do *software* bem como aumentar a comunicação entre os *designers*.

Para que a abordagem de padrões seja eficaz e evolua, cada *designer* que use os catálogos de padrões deve tentar optimizar os padrões existentes, de acordo com as suas próprias experiências de modelação, bem como derivar novos padrões e inseri-los num catálogo, logo que provem ser eficazes no processo de modelação do *designer*.

A cultura de partilha de dados através da Internet está espalhada pelos utilizadores SIG de todo o mundo. Este facto está a conduzir a uma grande proliferação de *sites* que oferecem algum tipo de dados interessantes. Isto deveu-se também, à criação de *clearinghouses* (organização central que recolhe e fornece informações a outras pessoas ou organizações), onde os metadados *standardizados* que descrevem conjuntos de dados geoespaciais são armazenados.

Também é defendido, neste artigo, que os *designers* de aplicações SIG devem ser apoiados por um sistema de catálogo de padrões distribuído por um sistema de *clearinghouse*, onde os padrões estão descritos e disponibilizados juntamente com os metadados relacionados com eles. Acredita-se também que os *designers* de aplicações SIG terão interesse em partilhar as suas experiências bem sucedidas na modelação de dados SIG como já ocorre na comunidade dos *designers* de orientação a objectos.

Quando os metadados e os padrões de análise estão integrados num sistema *clearinghouse* os metadados dos dados produzidos para uma aplicação podem fazer referência para o respectivo padrão que os descreve. Como consequência, o esquema do objecto do padrão pode ser usado para actualizar o esquema da Base de Dados do SIG em que os dados vão ser incorporados.

Com um mecanismo baseado em reutilização para suportar o desenvolvimento de aplicações SIG, todo o ciclo de desenvolvimento do *Software* pode tornar-se mais eficiente. Os padrões de análise podem permitir aos analistas e *designers* reutilizar o conhecimento dos profissionais experientes, sobretudo durante as fases iniciais do *design* dos SIG, evitando projectos de baixa qualidade devido à falta de experiência dos *designers* não especialistas.

Fica, assim, clara a importância da partilha das experiências no sentido de que o mesmo trabalho não tenha de ser feito inúmeras vezes por *designers* diferentes, permitindo que cada *designer* melhore o trabalho já feito em vez de ter de refazê-lo, o que mostra a importância do trabalho que se pretende fazer nesta dissertação, identificar alguns requisitos voláteis das aplicações SIG que sejam reutilizáveis, com vista a descrever os padrões que depois constituíram um catálogo de padrões de análise.

4.4. Abordagem para Desenhar Aplicações SIG, usando Padrões de Desenho

De acordo com Gordillo *et* al. (1999), para desenvolver uma aplicação geográfica, os *designers* têm de ter em conta diversos aspectos como a aquisição de dados, a precisão, a representação de relações espaciais, características topológicas e o desenho da *interface*. O problema destas aplicações é que não existe um método de desenho que consiga cobrir todos estes aspectos, o que se traduz em produtos finais com fraca capacidade de serem reutilizados, de serem modificados e com fraca modularidade (Gordillo *et* al., 1999).

As metodologias orientadas a objectos têm vindo a provar ser uma boa solução para o desenho de aplicações não convencionais. Assim, os autores optaram por usar esta metodologia, com vista a obter um *software* reutilizável, modular e modificável, bem como objectos que encapsulam conhecimento que pode ser adaptado a diferentes necessidades. Escolheram os padrões de desenho por ser uma estratégia poderosa para representar boas soluções de desenho para problemas recorrentes.

Na maioria das aplicações SIG, a dificuldade da implementação reside, sobretudo, na definição e na utilização da informação geográfica. Desta forma, neste artigo é proposto um processo de desenho iterativo com 2 passos, como forma de tornar esta tarefa mais simples e

clara. Este método leva a uma integração forte e transparente entre a informação espacial e a conceptual.

Durante a primeira etapa, a Modelação Conceptual, é descrito o domínio da aplicação em termos de entidades do mundo real. Isto é feito com a intenção de compreender o problema construindo uma representação abstracta na qual as características geográficas e os detalhes de implementação são ignorados. Nesta fase é usado o modelo orientado a objectos para representar a aplicação. Assim, o resultado deste passo não difere do que é obtido nas aplicações convencionais (as que não são SIG) com as técnicas de desenho orientado a objectos.

Este método permite manipular diferentes níveis de abstracção separando a compreensão do mundo descritivo do mundo geográfico. Acima de tudo, este processo também oferece uma forma flexível de estender as aplicações convencionais construídas com tecnologias orientadas a objectos, tirando-se assim proveito do modelo conceptual que acabou de ser definido e pensando em adicionar características espaciais em vez de redefinir o sistema todo.

No passo seguinte o objectivo é definir as características espaciais para as classes definidas no modelo conceptual. É proposto que o modelo conceptual seja enriquecido em vez de modificado. Começa-se por identificar quais as classes do modelo conceptual que têm características espaciais. Feito isto, para cada uma dessas classes é definida uma nova classe que a envolva com comportamento espacial aplicando o padrão de desenho *decorator*.

Um *decorator* atribui responsabilidades adicionais a um objecto de forma dinâmica. Assim, oferece uma alternativa flexível para efectuar *subclassing* de forma a estender uma funcionalidade de uma classe. A principal diferença entre *decorating* e *subclassing* é que enquanto a primeira é dinâmica a segunda é estática. Usando *decorators* torna-se possível adicionar funcionalidades a alguns objectos sem redefinir a hierarquia conceptual.

Na abordagem proposta, a ideia deste padrão de desenho é usada para obter o modelo geográfico básico, sendo que depois se vão adicionando características espaciais a cada objecto de forma dinâmica e transparente. Este esquema é útil como ferramenta conceptual para novos desenhos, mas também para alterar desenhos existentes com vista a incluir informação geográfica.

O trabalho de Gordillo *et* al. (1999) trata-se de uma técnica de modelação de SIG orientada a objectos. Nesta abordagem, obtém-se o modelo geográfico básico, sendo que depois se vão

adicionando características espaciais a cada objecto de forma dinâmica e transparente. Desta forma, mostra-se como pode ser útil a reutilização neste tipo de aplicações. Este trabalho ilustra também algo semelhante ao que se pretende fazer no caso de estudo, isto é, pretende-se que se vão aplicando ao sistema CLIP[4] alguns padrões, o que se traduz na adição de algumas características SIG voláteis à aplicação.

4.5. OMT-G (Object Modelling Techniques for Geographic Applications)

Devido às características peculiares dos dados geográficos é fundamental que haja uma definição mais precisa dos objectos, operações, e parâmetros de visualização necessários, isto obtém-se através da aproximação do desenho conceptual e da implementação das aplicações geográficas, que se atinge usando 3 diagramas diferentes, um para especificar classes de objectos e as suas relações; outro para especificar operações de transformação entre as classes; e um para especificar os diversos aspectos visuais que cada classe de objectos pode assumir, como é pedido pela aplicação. Assim, o modelo OMT-G (*Object Modelling Techniques for Geographic Applications*) proposto por Borges *et* al. (2001) é baseado principalmente em 3 conceitos: classes; relações e restrições de integridade espacial, portanto propõem a utilização de 3 diagramas diferentes no processo de desenho de uma aplicação geográfica. O primeiro diagrama é o habitual diagrama de classes, no qual todas as classes são especificadas juntamente com as suas representações e relações. A partir deste diagrama é possível derivar um conjunto de restrições de integridade espacial que devem ser observadas na implementação. Quando o diagrama de classes indica a necessidade de múltiplas representações de qualquer classe, ou quando a aplicação envolve a derivação de alguma classe a partir de outras, deve ser construído o diagrama de transformação. Neste, todos os processos de transformação podem ser especificados, permitindo a identificação de todos os métodos necessários para a implementação. Finalmente, deve ser construído um diagrama de apresentação para dar as orientações relativas ao aspecto visual dos objectos na implementação. Podem existir diversos aspectos visuais para qualquer uma das classes, o que permite a definição de uma vista ou conjuntos de vistas para cada aplicação ou grupo de utilizadores. A identificação das restrições de integridade espacial é uma actividade importante do esquema do desenho da base de dados de determinada aplicação e envolve a identificação das restrições de integridade que devem ser mantidas na base de dados.

[4] http://clip.unl.pt

Devido ao recurso que o OMT-G faz dos diagramas para representar a geometria das classes de objectos georreferenciados, o esquema construído através desta técnica é mais compacto, intuitivo e mais perceptível que aqueles derivados de modelos que descrevem os tipos geométricos através de relações. Na verdade, o OMT-G apresenta vantagens muito claras quando aplicado na modelação de diversos tipos de aplicações, como as relacionadas com o ambiente, com os sistemas urbanos e com a cartografia automática. No estudo feito com o modelo OMT-G, ficou provado que este é capaz de representar aspectos particulares dos dados geográficos relativos ao campo da aplicação, enquanto preserva a clareza e a facilidade de representação.

No seu trabalho, Borges *et* al. (2001) propõem um modelo baseado no diagrama de classes, no de transformações e no de apresentação. Mostra-se como o recurso aos diagramas para representar as classes de objectos georreferenciados é uma mais-valia, na medida em que o esquema que se obtém através desta técnica é mais compacto, intuitivo e perceptível do que aqueles derivados de modelos que descrevem os tipos geométricos através de relações. Fica, assim, claro que é possível representar aspectos particulares dos dados geográficos preservando a clareza e a facilidade de representação.

4.6. Sumário

Através dos trabalhos relacionados que foram apresentados acima, podemos ver que já se fazem alguns trabalhos relacionados com a modelação comportamental e de dados para SIG. No primeiro trabalho (Oliveira, 2009) é feita a modelação de SIGs usando aspectos, obtendo-se um modelo que consiste na identificação, modularização e composição dos *crosscutting concerns*, mais precisamente dos *concerns* espaciais. Ora nesta dissertação pretende-se fazer o mesmo mas os *concerns* a identificar serão os *concerns* espaciais voláteis.

No trabalho seguinte, Zipf *et* al. (2003) consideram que a programação orientada a aspectos ajuda na resolução de alguns problemas de manutenção e reutilização dos SIG. Desta forma, é fundamental que seja feita uma boa análise de domínio dos SIG, para se conseguirem identificar os padrões de reutilização.

No seu trabalho, Lisboa *et* al. (1998) defendem que a abordagem de Padrões pode ser muito benéfica para o domínio SIG. Permitindo a criação de um catálogo de padrões distribuído, gerido pelos designers SIG. Tornando-se evidente a importância que a partilha de experiências representa para que o trabalho dos designers possa ser melhor.

O trabalho de Gordillo *et* al. (1999) consiste numa técnica de modelação de SIG orientada a objectos, na qual se obtém um modelo geográfico básico, onde posteriormente, de forma dinâmica e clara, se vão acrescentando características espaciais aos objectos. Mostrando-se, assim, como a reutilização é importante para os SIG.

Finalmente, no último, (Borges *et* al., 2001) propõe-se um modelo baseado no diagrama de classes, no de transformações e no de apresentação. Mostrando como se pode representar características dos dados geográficos mantendo a representação clara e simples.

5. Definição dos Padrões Geoespaciais

Neste capítulo vai ser apresentado o *template* usado para a definição dos padrões geoespaciais. Sendo que, posteriormente, se apresentam as descrições de dois exemplos de padrões geoespaciais seleccionados usando esse mesmo *template* para os descrever.

5.1. Definição dos Padrões

Nesta secção apresentam-se as descrições dos padrões de *Localizar Espacialmente uma Entidade de uma Aplicação sem Georreferenciação* e *Apresentação de Entidades Georreferenciadas*. Estes dois padrões foram identificados através da análise de domínio efectuada para SIGs (Urbieta *et* al., 2010). Além dos dois padrões seleccionados para serem descritos nesta dissertação foram identificados mais alguns padrões, como é o caso de *Adicionar Disponibilidade Temporal aos Objectos, Manutenção de Perfil e Ajustar Conjunto de Dados Espaciais.* Os padrões descritos nesta tese foram dois dos que foram considerados mais importantes e mais elucidativos para explicar a técnica. Isto porque, para podermos aplicar qualquer um dos restantes padrões, era necessário localizar o objecto espacialmente, e para que ele seja visto é necessário que este seja apresentado visualmente.

O *template* usado para definir os padrões – Tabela 5.1 – será baseado no *template* sugerido por Pantoquilho *et* al. (2003). Em que se começa por apresentar o nome do padrão, seguido da descrição do problema que este pretende resolver, o contexto em que o problema se insere e a aplicabilidade deste padrão. De seguida são apresentados os requisitos funcionais e não-funcionais deste padrão, bem como as dependências entre esses requisitos, as prioridades e os conflitos existentes entre eles. Depois, é feita a modelação do padrão através de um diagrama de *features*, de um diagrama de classes e de um diagrama de sequência. Finalmente, são apresentados exemplos onde este padrão poderá ser aplicado, as consequências da sua aplicação e os padrões que se relacionam com ele.

Tabela 5.1. *Template* para especificar os padrões geoespaciais

Campo			**Descrição**
Nome			Nome identificador do Padrão.
Problema			Descreve o problema que o padrão pretende solucionar.
Contexto			Descreve o ambiente em que o problema e a solução se aplicam.
Requisitos	Funcionais		Lista dos requisitos funcionais do padrão.
	Não-Funcionais		Lista dos requisitos não-funcionais do padrão.
	Dependências		Identificação das dependências entre os requisitos funcionais e não-funcionais.
	Actores		Identifica os actores do padrão.
Modelação	Estrutural	Diagrama de *Features*	Diagrama de *Features* do Padrão.
		Diagrama de Classes	Diagrama de Classes do Padrão.
	Comportamental	Diagrama de Sequência	Diagrama de Sequência do Padrão.
Consequências			Vantagens e desvantagens da aplicação do padrão.
Lista de Eventos			Identifica os eventos que despoletam o padrão.
Exemplos	Diagrama de *Features*		Exemplos que ilustram o contexto do Padrão, como é aplicado e as alterações necessárias ao contexto inicial.
	Diagrama de Classes		
	Diagrama de Sequência		
Padrões Relacionados			Lista de Padrões que se relacionam com o presente padrão.

Para a criação dos padrões começou por se fazer uma análise das aplicações de Sistemas de Informação Geográfica com vista a identificar *concerns* voláteis que fossem passíveis de reutilização. Feito isto, identificaram-se os padrões de análise e depois fez-se a descrição de cada um deles por meio do *template* apresentado acima, na tabela 5.1.

Assim, para descrever um padrão é necessário começar por escolher o nome adequado, isto é, um nome elucidativo do que o padrão pretende resolver. Feito isto, é necessário descrever qual o problema que o padrão proposto pretende resolver e depois explica-se o contexto em que este estará inserido, ou seja, para que tipo de situações ele está direccionado. Depois de introduzido o propósito do padrão é necessário fazer a descrição dos requisitos do mesmo.

Esta descrição consiste na enumeração dos requisitos funcionais e não-funcionais, seguindo-se a identificação das dependências existentes entre eles e depois a identificação dos actores que irão ter intervenção no problema.

Depois de assentes os requisitos do padrão procedeu-se à modelação do mesmo. A modelação teve diversas etapas, começando pela modelação estrutural, em que há a preocupação de descrever a estrutura da solução, isto é, que características deverão estar presentes na solução e que classes vão ser necessárias na implementação da solução. Esta modelação será feita por meio de um diagrama de *Features*, onde se vão identificar as características do problema e por meio de um diagrama de Classes, onde se vão identificar as classes e as suas hierarquias, que serão necessárias para a implementação da solução. Segue-se a modelação comportamental, em que há a preocupação de descrever o comportamento que a solução deve ter, ou seja, que actividades vão ser necessárias para resolver o problema em causa. A modelação comportamental será feita através de um diagrama de sequência, sendo que se irá recorrer a metodologias DSOA para facilitar a modularização. Mais concretamente, foram usados cenários aspectuais e composição, realizada através dos mecanismos da abordagem MATA. Assim, o diagrama de sequência irá descrever o aspecto que representa o comportamento do padrão, e nos exemplos será feita a composição do cenário base com o cenário aspectual.

Após terminada a modelação procedeu-se à descrição das consequências da aplicação do padrão, nomeadamente apresentam-se os pontos positivos e as limitações da aplicação do mesmo.

Foi ainda feito o levantamento de alguns eventos que seriam passíveis de despoletar o padrão em causa, ou seja, situações em que seria benéfica a utilização do padrão.

Depois de descritos os pormenores do padrão é feita a sua aplicação a título de exemplo. Para descrever os exemplos são redefinidos os diagramas de *features*, classes e de sequência para ficarem de acordo como o caso do exemplo. Como a modelação do padrão corresponde ao aspecto, os exemplos irão incluir os diagramas para os cenários base e composto.

Depois de todo o padrão descrito é feita uma análise, com vista a identificar possíveis relações entre o presente padrão e outros padrões, de que resulta uma lista de padrões relacionados.

Findo todo este processo, tem-se a descrição detalhada do padrão, permitindo que este venha a ser reutilizado mais tarde noutras aplicações.

5.1.1. Padrão de *Localizar Espacialmente uma Entidade de uma Aplicação sem Georreferenciação*

Nome: Localizar Espacialmente uma Entidade de uma Aplicação sem Georreferenciação.

Problema: Muitas aplicações *Web* não foram concebidas com a intenção de disponibilizar características geoespaciais. No entanto, para que suportem determinados comportamentos espaciais tem de se adicionar algum comportamento deste tipo às entidades que compões estas aplicações.

Contexto: Este problema aplica-se a sistemas que não possuam características geoespaciais, mas que podem beneficiar delas. Tem-se o exemplo de uma aplicação de uma empresa de transportes por autocarro que disponibiliza os horários das carreiras, podendo fornecer, também, a sua localização em tempo real e informações mais precisas e atempadas para os gestores do sistema e passageiros.

Requisitos:

- Funcionais:

1. Obter uma entidade localizável de uma aplicação sem georreferenciação;
2. Verificar se a entidade é estática ou móvel:
    2.1. Se for estática, obter os seus dados de localização;
    2.2. Se for móvel:
        2.2.1. Obter hora do sistema;
        2.2.2. Obter o horário associado à entidade;
        2.2.3. Verificar no horário se a entidade está na localização secundária;
            2.2.3.1. Se não estiver na localização secundária, verificar se a entidade está em hora de trabalho;
                2.2.3.1.1. Se não estiver, devolve mensagem de erro;
                2.2.3.1.2. Se estiver, obter a localização principal;
            2.2.3.2. Se estiver na localização secundária, obter esta localização.

- Não-Funcionais:

1. Correcção:

    1.1. Na obtenção da entidade;

    1.2. Na verificação de se a entidade é estática ou móvel;

    1.3. Na obtenção dos dados de localização da entidade estática;

    1.4. Na obtenção da hora do sistema

    1.5. Na obtenção do horário associado à entidade;

    1.6. Na verificação de se a entidade está na localização secundária;

    1.7. Na verificação de se a entidade está na hora de trabalho;

    1.8. Na obtenção da localização principal;

    1.9. Na obtenção da localização secundária.

2. Tempo de resposta:

    2.1. Na verificação de se a entidade é estática ou móvel;

    2.2. Na obtenção dos dados de localização da entidade estática;

    2.3. Na obtenção da hora do sistema;

    2.4. Na obtenção do horário associado à entidade;

    2.5. Na verificação de se a entidade está na localização secundária;

    2.6. Na verificação de se a entidade está na hora de trabalho;

    2.7. Na obtenção da localização principal;

    2.8. Na obtenção da localização secundária.

3. Precisão:

    3.1. Na obtenção dos dados de localização da entidade estática;

    3.2. Na obtenção da hora do sistema;

3.3. Na obtenção do horário associado à entidade;

3.4. Na obtenção da localização principal;

3.5. Na obtenção da localização secundária.

- Dependências:

Requisito Funcional	Depende de	Requisito Não-Funcional
RF 1	→	RNF 1.1
RF 2	→	RNF 1.2
		RNF 2.1
RF 2.1	→	RNF 1.3
		RNF 2.2
		RNF 3.1
RF 2.2.1	→	RNF 1.4
		RNF 2.3
		RNF 3.2
RF 2.2.2	→	RNF 1.5
		RNF 2.4
		RNF 3.3
RF 2.2.3	→	RNF 1.6
		RNF 2.5
RF 2.2.3.1	→	RNF 1.7
		RNF 2.6
RF 2.2.3.1.2	→	RNF 1.8
		RNF 2.7
		RNF 3.4
RF 2.2.3.2	→	RNF 1.9
		RNF 2.8
		RNF 3.5

- Actores:
  - Utilizador; Entidades; Sistema/Aplicação;

Modelação:

- **Estrutural:**
  - **Diagrama de *Features*:**

No diagrama abaixo – Figura 5.1 –, temos as características necessárias para localizar uma entidade sem georreferenciação – *Locate an Entity without Georeferencing*. Temos que é obrigatório haver uma *Locatable Entity* que pode ser *Static* ou *Mobile*, não podendo ser as duas ao mesmo tempo. Tem de haver também, obrigatoriamente, uma *Location*, sendo que esta tem de ter uma *Longitude* e uma *Latitude* e pode ter ou não uma *Altitude*. Opcionalmente podemos ter ainda um *Schedule*. Quando temos uma *Mobile Entity* é necessário que haja uma

Static Entity e um *Schedule*. A vantagem deste diagrama é identificar a variabilidade do padrão.

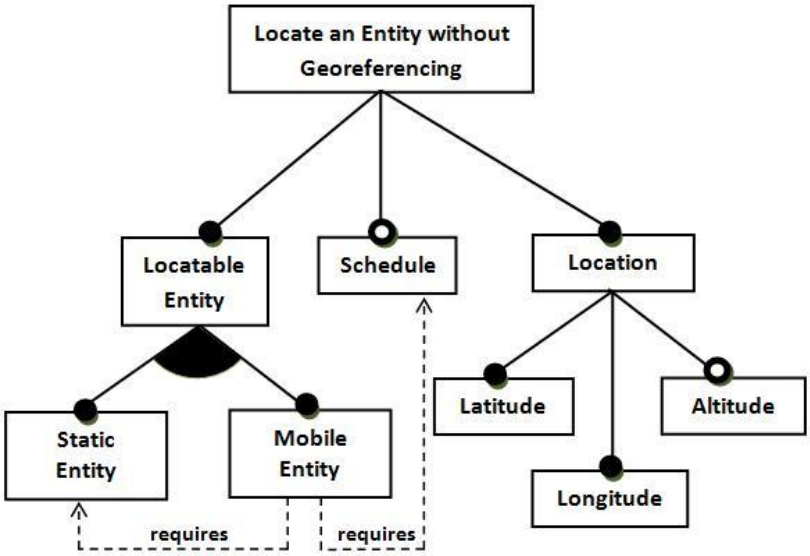

Figura 5.1. Diagrama de *Features* para Localizar Entidade

- **Diagrama de Classes:**

No diagrama abaixo – Figura 5.2 –, temos 6 classes. A classe de *Interface*, a de *Control*, a de *Locatable Entity* e temos ainda as classes *Schedule*, *Main Location* e *Secondary Location*. A classe de *Interface* é a intermediária entre o utilizador e o *Control*, e esta última – a classe de *Control* – é a responsável por fazer as verificações e pedir os dados necessários. A classe de *Control* comunica com a classe *Locatable Entity*. Esta – a *Locatable Entity* –, como se pode ver no diagrama, tem um único *Schedule*, uma *Main Location* e pode ter várias *Secondary Location*. Um *Schedule* pertence apenas a uma *Locatable Entity*. Tanto uma *Main Location* como uma *Secondary Location* podem ser de várias *Locatable Entity*.

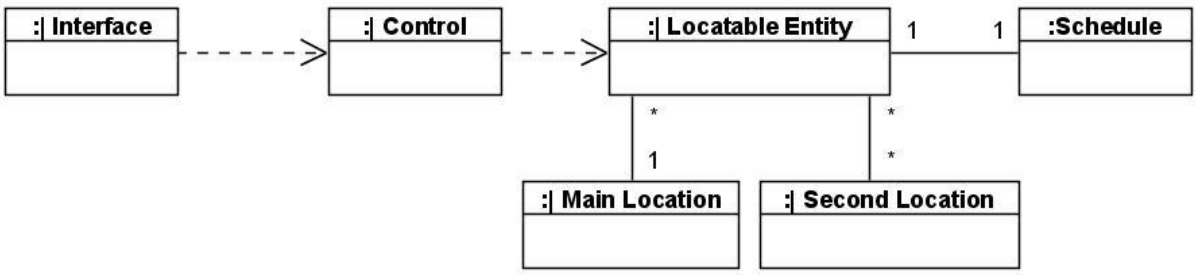

Figura 5.2. Diagrama de Classes para Localizar Entidade

- **Comportamental – Diagrama de Sequência (Figura 5.3):**

O diagrama de Sequência apresentado na Figura 5.3 corresponde ao cenário aspectual do padrão. Este diagrama descreve o cenário onde é obtida a localização de uma entidade (o aspecto). Como podemos ver, esta tem início no *Control* que solicita a informação relativa à *Locatable Entity* e depois verifica se esta é *Static* ou *Mobile*. Se a *Locatable Entity* for *Static,* o *Control* informa a *Interface* que obtém os dados de localização da mesma. Caso seja *Mobile* o *Control* também informa a *Interface* e este obtém a hora do sistema e solicita, ao *Control,* o *Schedule* associado à *Locatable Entity,* o *Control* solicita o *Schedule* da *Locatable Entity* à entidade *Schedule* e verifica se, no momento, a *Locatable Entity* está na *Secondary Location* e informa a *Interface*. Se não estiver, a *Interface* solicita ao *Control* a verificação de se a *Locatable Entity* está em *working hour* (dentro do horário de trabalho). Se não estiver, a *Interface* é informada e envia uma mensagem ao Utilizador, a informar de que não e possível obter a localização da *Locatable Entity*. Se estiver em *working hour*, o *Control* informa a *Interface* e solicita à entidade *Main Location* a sua localização. Se esta for a *Secondary Location,* o *Control* informa a *Interface* e solicita à entidade *Secondary Location* a sua localização.

Consequências: A aplicação deste padrão irá permitir que se localize uma entidade de uma aplicação espacialmente, permitindo que esta possa ser, posteriormente apresentada visualmente, por exemplo, num mapa.

Lista de Eventos:

1. Ocorre um incêndio num edifício e é necessário localizar todos os ocupantes do mesmo para garantir que todos saem do edifício;

2. Uma empresa de transportes por autocarro necessita indicar, aos utentes, a localização de determinado autocarro em determinado momento e, para esse efeito necessita de executar a sua localização;

3. Uma empresa que fornece serviços técnicos necessita de um sistema que lhe permita localizar cada técnico a prestar assistência, de forma a poder gerir melhor o trabalho de cada um.

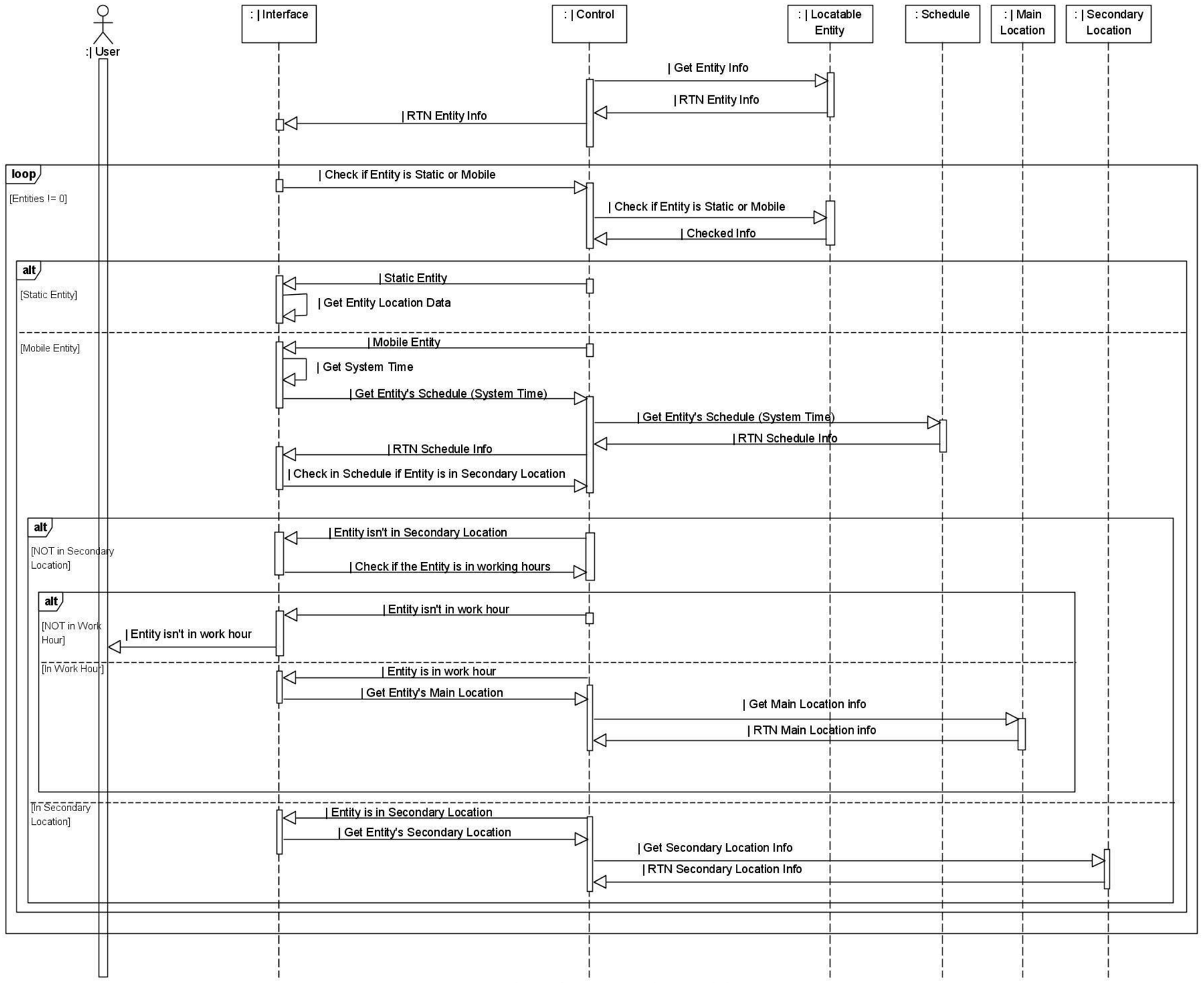

Figura 5.3. Diagrama de Sequência para Localizar Entidade

Exemplos:

Este padrão pode ser aplicado a entidades estáticas, isto é, entidades que têm sempre a mesma localização e, neste caso, esta é simplesmente armazenada em Base de Dados. Por outro lado, o padrão pode também ser aplicado a entidades móveis, ou seja, entidades que mudam de localização de acordo com determinadas características, como é o caso de um indivíduo, ou de um veículo.

Como exemplo, vamos analisar a aplicação deste padrão ao sistema CLIP, o sistema de informação das disciplinas e horários de todos os alunos, docentes e restantes funcionários e colaboradores da Universidade Nova de Lisboa.

Vamos observar um exemplo de localização de uma entidade estática e de localização de uma entidade móvel.

Como exemplo de localização de entidade móvel, temos:

Acontece uma urgência e é necessário convocar uma reunião de professores. Assim, é necessário localizá-los no Campus.

Localizar espacialmente a Entidade Docente:

Modelação:

- **Diagrama de Features:**

No diagrama abaixo – Figura 5.4 –, temos as características necessárias para localizar um docente – *Locate Spatially Lecturer Entity*. Como se vê, é necessário ter um *Lecturer*, o seu *Schedule* e a sua *Location*, sendo que esta última tem de ser dada por uma *Latitude* e uma *Longitude*, podendo também haver uma *Altitude*. Poderá haver também o *Office* do docente e uma *Room*.

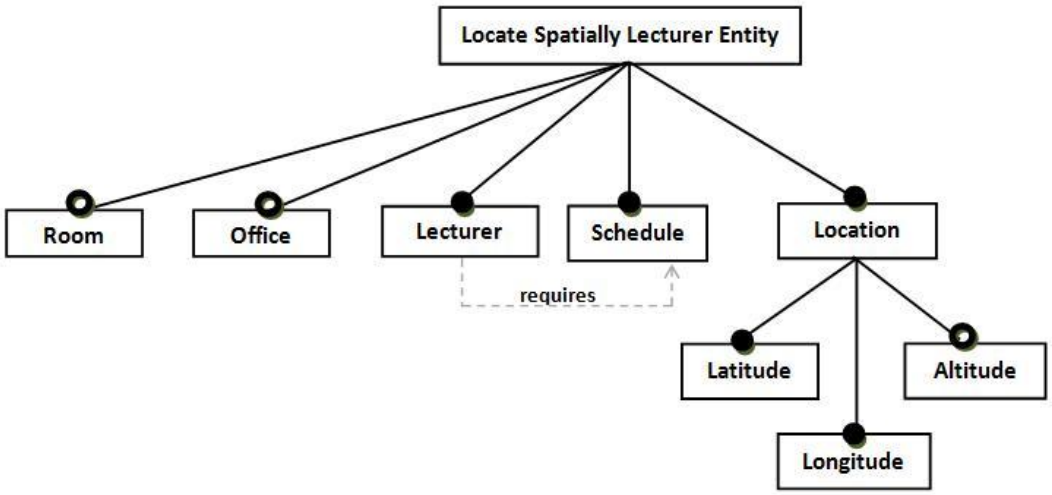

Figura 5.4. Diagrama de *Features* para Localizar Docente

- **Diagrama de Classes:**

Na Figura 5.5 temos o diagrama de classes para o cenário base do exemplo, este cenário corresponde à localização dos docentes convocados para uma reunião. Assim, serão necessárias as classes *CLIP Interface*, *CLIP Control*, *Lecturer* e *Meeting*. Na Figura 5.6 temos o diagrama de classes para o cenário composto da base com o aspecto. Portanto, temos as classes do Aspecto instanciadas com as do exemplo e acrescentamos as da base. Temos, assim as classes do Aspecto instanciadas, *Interface* com *CLIP Interface*, *Control* com *CLIP Control*, *Locatable Entity* com *Lecturer*, *Main Location* com *Office* e *Secondary Location* com *Class Room*. A classe *Schedule* não precisa de ser instanciada e no cenário base acresce a classe *Meeting*.

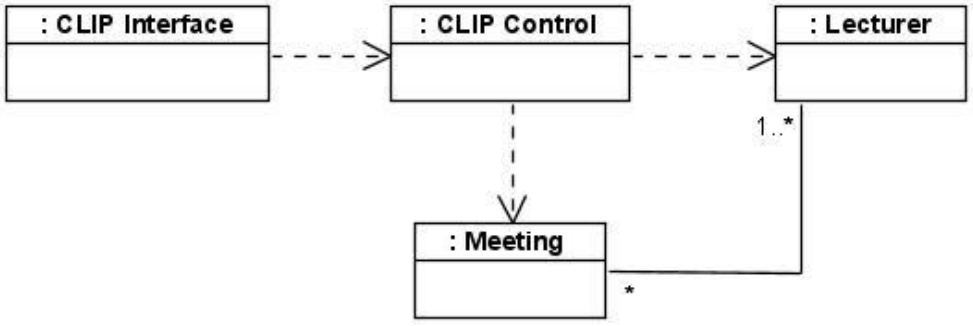

Figura 5.5. Diagrama de Classes do Cenário Base para Localizar Docente

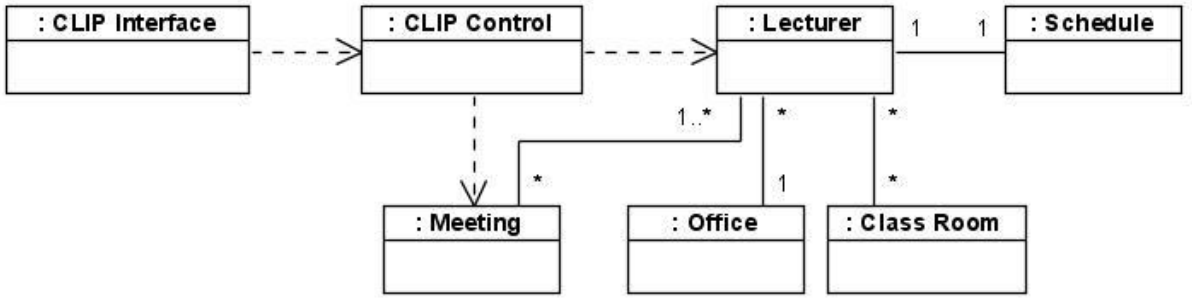

Figura 5.6. Diagrama de Classes do Cenário Composto para Localizar Docente

- **Diagrama de Sequência:**

Na Figura 5.7 temos o diagrama de sequência para o cenário base do exemplo. Como já foi dito, este cenário corresponde à localização dos docentes para uma reunião. É iniciado com o pedido de contacto dos docentes convocados para uma reunião por parte do Utilizador. A *Interface* solicita ao *Control* a informação da *Meeting*, sendo que este faz o pedido da informação à entidade *Meeting*. A informação que a entidade *Meeting* devolve ao *Control* inclui a lista dos docentes que foram convocados para a mesma. Com esta informação o *Control* pede à entidade *Lecturer* a informação sobre cada docente e depois envia esta informação para a *Interface*, sendo depois a *Interface* a devolver a mesma ao Utilizador.

Na Figura 5.8, temos o diagrama de sequência para o cenário composto da base com o aspecto, e portanto temos a composição dos fluxos da base com os do aspecto. As entidades do Aspecto são instanciadas com as do exemplo ao que se acrescenta as da base. As entidades do Aspecto instanciadas são *Interface* com *CLIP Interface*, *Control* com *CLIP Control*, *Locatable Entity* com *Lecturer*, *Main Location* com *Office* e *Secondary Location* com *Class Room*. A classe *Schedule* não precisa de ser instanciada e do cenário base acresce a classe *Meeting*. Neste diagrama temos, no fundo, o cenário base até ao momento em que a informação do docente é devolvida ao *CLIP Control*, sendo que depois disto temos o cenário aspectual – apresentado na Figura 5.3 – voltando-se depois, no final do cenário Aspectual, ao cenário base.

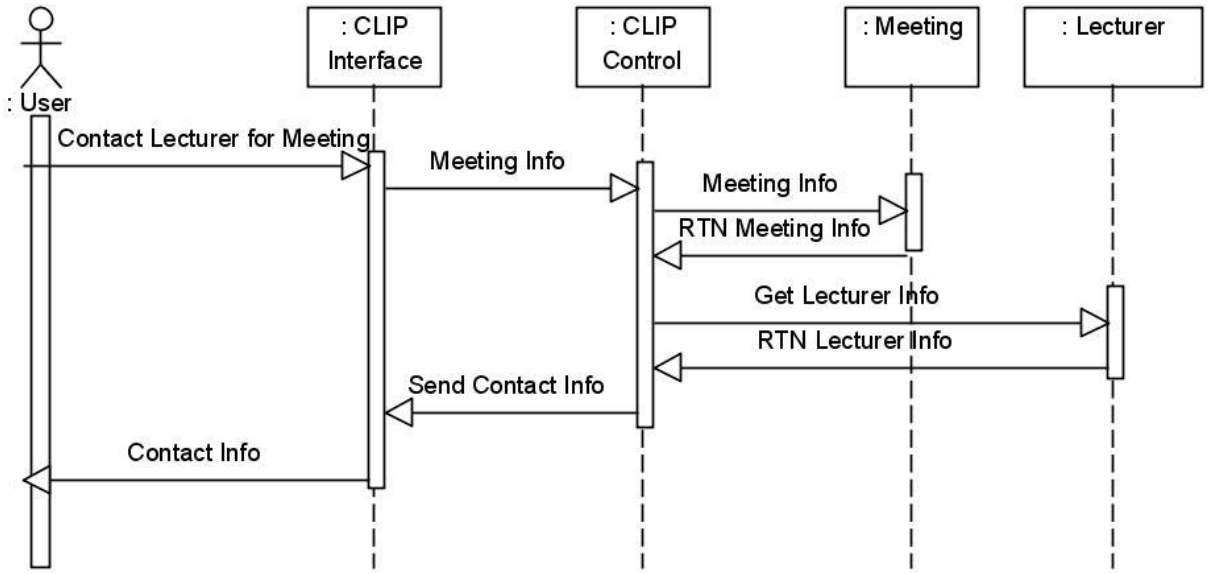

Figura 5.7. Diagrama de Sequência do Cenário Base para Localizar Docente

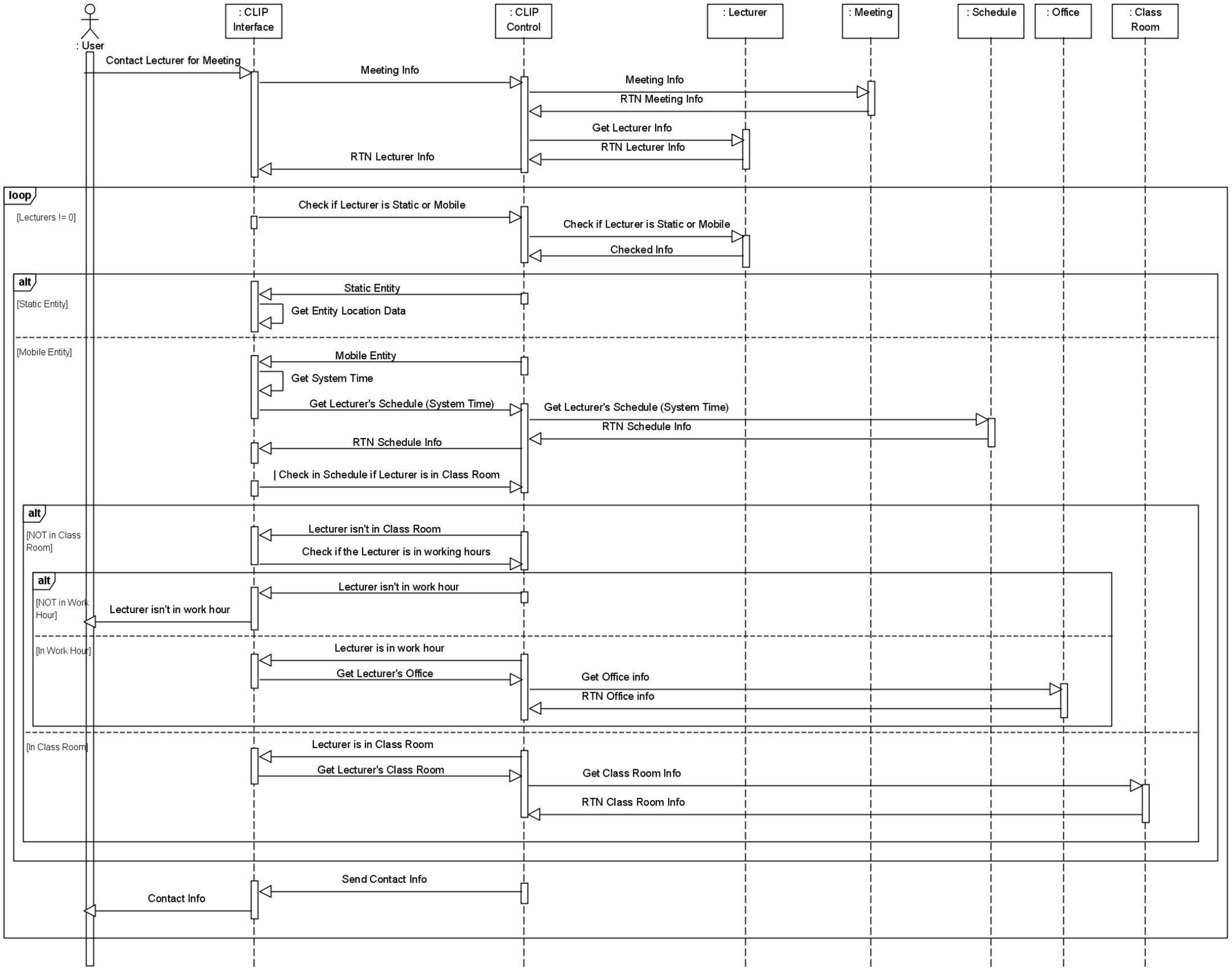

Figura 5.8.Diagrama de Sequência do Cenário Composto para Localizar Docente

Como exemplo de localização de entidade estática, temos:

Um professor faz parte de um júri de uma dissertação de Mestrado, e precisa de saber onde a defesa da mesma, a que tem de assistir, vai decorrer. Assim, é necessário obter a localização da sala em causa.

Localizar espacialmente a Entidade Sala:

Modelação:

- **Diagrama de Features:**

No diagrama abaixo – Figura 5.9 –, temos as características necessárias para localizar uma sala de aula – *Locate Spatially Room Entity*. Como se vê, é necessário ter uma *Room* e a sua *Location*, sendo que esta última tem de ser dada por uma *Latitude* e uma *Longitude*, podendo também haver uma *Altitude*.

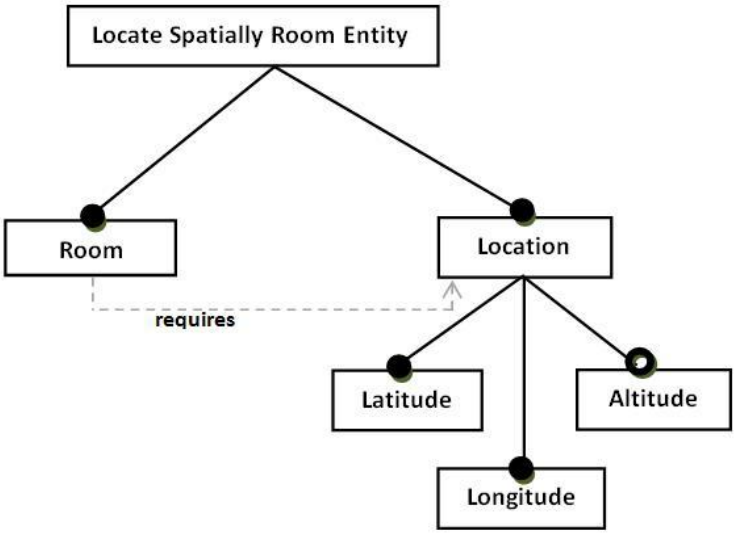

Figura 5.9. Diagrama de *Features* para Localizar Sala

- **Diagrama de Classes:**

Na Figura 5.10 temos o diagrama de classes para o cenário base do exemplo. Este cenário corresponde à localização da sala de aula onde vai decorrer a discussão de uma tese. Assim, serão necessárias as classes *CLIP Interface*, *CLIP Control* e *Thesis Discussion*. Na Figura 5.11 temos o diagrama de classes para o cenário composto da base com o aspecto. Desta forma, temos as classes do Aspecto instanciadas com as do exemplo e acrescentamos as da base. Assim, as classes do Aspecto instanciadas são: *Interface* com *CLIP Interface*, *Control* com *CLIP Control*, *Locatable Entity* com *Class Room*, *Main Location* e *Secondary Location*

não estão instanciadas porque não são necessárias para o exemplo em causa, pois a entidade *Class Room* trata-se de uma entidade estática. A classe *Schedule* não precisa de ser instanciada e do cenário base acresce a classe *Thesis Discussion*.

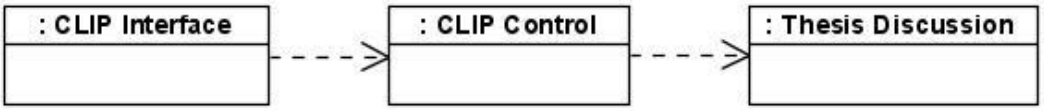

Figura 5.10. Diagrama de Classes do Cenário Base para Localizar Sala de Aula

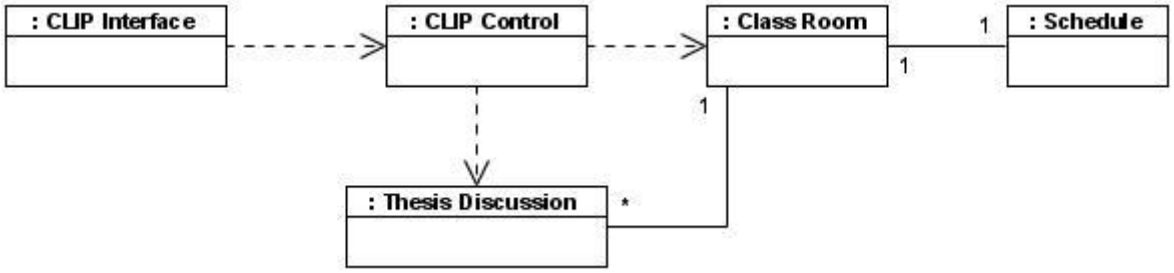

Figura 5.11. Diagrama de Classes do Cenário Composto para Localizar Sala de Aula

- **Diagrama de Sequência:**

Na Figura 5.12 temos o diagrama de sequência para o cenário base do exemplo. Como já foi dito, este cenário corresponde à localização de uma Sala de aula onde vai decorrer a discussão de uma tese. É iniciado com o pedido de localização da discussão da tese por parte do Utilizador. A *Interface* solicita ao *Control* a informação da Discussão da Tese, sendo que este faz o pedido da informação à entidade *Thesis Discussion*. A informação que a *Thesis Discussion* devolve ao *Control* inclui a identificação da Sala de Aula onde vai decorrer a discussão. Feito isto, o *Control* envia a informação para a *Interface*, sendo depois esta a devolvê-la ao Utilizador.

Na Figura 5.13, temos o diagrama de sequência para o cenário composto da base com o aspecto, e portanto temos a composição dos fluxos da base com os do aspecto. Desta forma, as entidades do Aspecto são instanciadas com as do exemplo e acrescentamos as da base. Assim, as entidades do Aspecto instanciadas são: *Interface* com *CLIP Interface*; *Control* com *CLIP Control*; e *Locatable Entity* com *Class Room*. A *Main Location* e a *Secondary Location* não são instanciadas porque, como já foi dito, a *Class Room* é uma entidade estática. A classe *Schedule* não precisa de ser instanciada e do cenário base acresce a classe *Thesis Discussion*. Neste diagrama temos, no fundo, o cenário base até ao momento em que a informação da

discussão é devolvida ao *Control*, sendo que depois disto temos o cenário aspectual – apresentado na Figura 5.3 – voltando-se depois, no final do cenário Aspectual, ao cenário base.

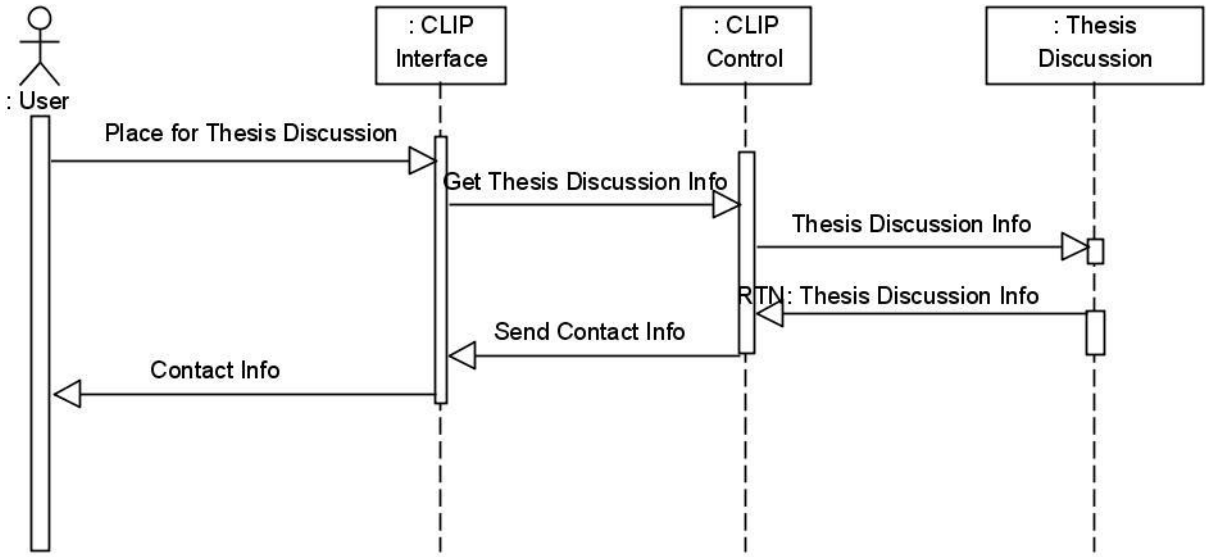

Figura 5.12. Diagrama de Sequência do Cenário Base para Localizar Sala

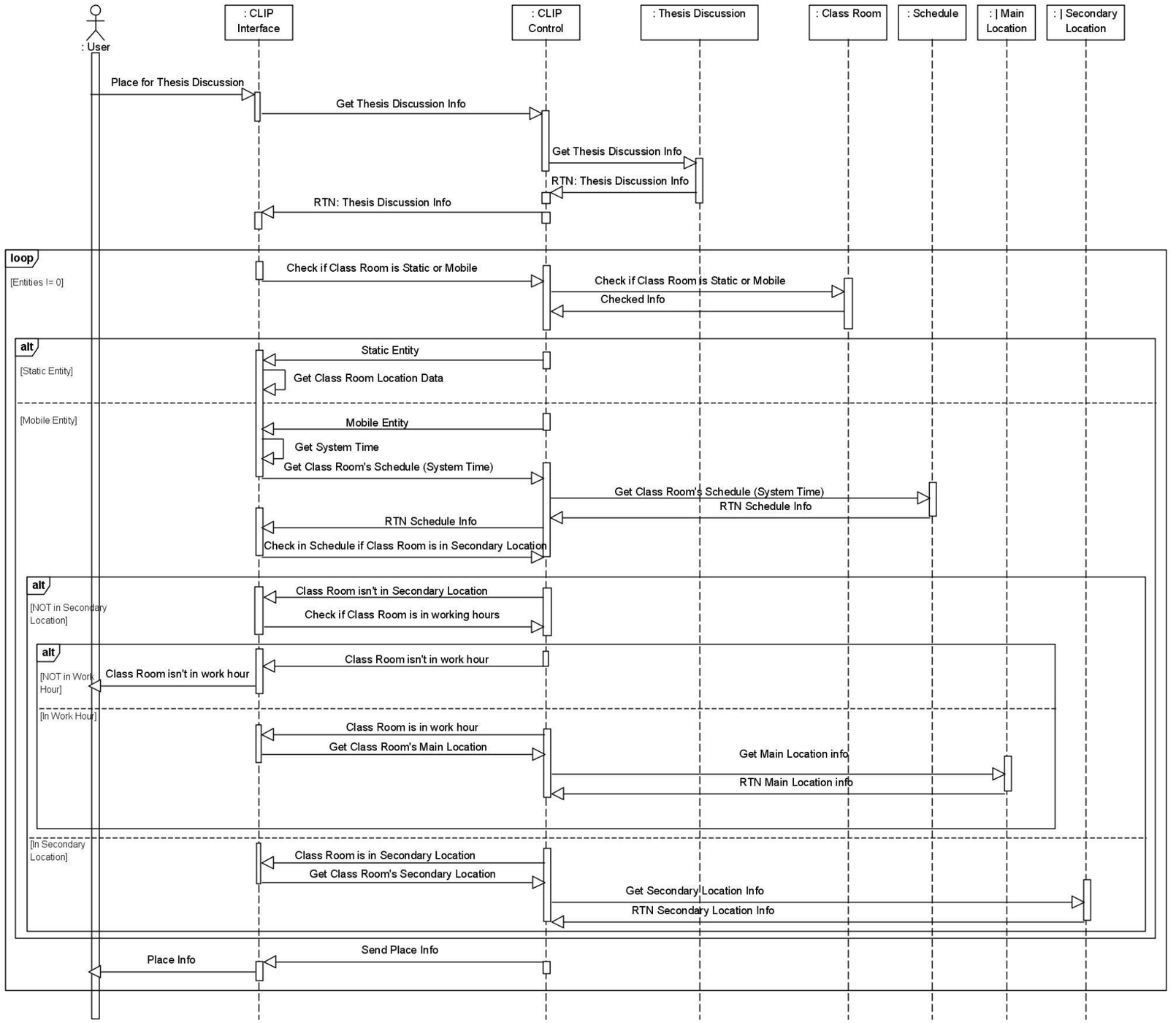

Figura 5.13. Diagrama de Sequência do Cenário Composto para Localizar Sala

Padrões Relacionados (Urbieta, 2010):

- Adicionar disponibilidade temporal às entidades espaciais;
- Apresentação das entidades espaciais;
- Ajustar o estado das entidades (Adicionar/Remover todas ou parte das entidades espaciais).

5.1.2. Padrão de *Apresentação de Entidades Georreferenciadas*

Nome: Apresentação de Entidades georreferenciadas.

Problema: Para que uma aplicação *Web* dotada de georreferenciação tenha efeito para o utilizador final é necessário que as entidades georreferenciadas sejam apresentadas visualmente.

Contexto: Este problema aplica-se a objectos que possuam características geoespaciais, mas não tenham uma apresentação visual.

Requisitos:

- Funcionais:

1. Obter uma entidade georreferenciada;
2. Verificar se a entidade é estático ou móvel:
    2.1. Se for estático, obter os seus dados de localização;
    2.2. Se for móvel:
        2.2.1. Obter hora do sistema;
        2.2.2. Obter o horário associado à entidade;
        2.2.3. Verificar no horário se a entidade está na localização secundária:
            2.2.3.1. Se não estiver na localização secundária, verificar se a entidade está em hora de trabalho:
                2.2.3.1.1. Se não estiver, devolve mensagem de erro;
                2.2.3.1.2. Se estiver, obter a localização principal;

2.2.3.2. Se estiver na localização secundárias, obter esta localização;

3. Obter a representação gráfica da entidade (poderá ser uma representação XML).
4. Verificar se existe alguma restrição/informação para adicionar à entidade:
   4.1. Se existir, adicionam-se as restrições/informações relativas à representação da entidade;
5. Obter mapa;
6. Sobrepor a representação gráfica da entidade ao mapa.

- Não-Funcionais:

1. Correcção:
   1.1. Na obtenção da entidade;
   1.2. Na verificação de se o objecto é estático ou móvel;
   1.3. Na obtenção da localização da entidade estática;
   1.4. Na obtenção da hora do sistema;
   1.5. Na obtenção do horário associado à entidade;
   1.6. Na verificação de se a entidade está na localização secundária;
   1.7. Na verificação de se a entidade está na hora de trabalho;
   1.8. Na obtenção da localização principal;
   1.9. Na obtenção da localização secundária;
   1.10. Na obtenção da representação gráfica da entidade;
   1.11. Na verificação e obtenção das restrições;
   1.12. Na sobreposição com o mapa;
2. Tempo de resposta:
   2.1. Na obtenção da localização da entidade estática;

2.2. Na obtenção da hora do sistema;

2.3. Na obtenção do horário associado à entidade;

2.4. Na obtenção da localização principal;

2.5. Na obtenção da localização secundária;

2.6. Na obtenção das restrições;

2.7. Na obtenção do mapa;

2.8. Na sobreposição com o mapa;

3. Precisão:

3.1. Na obtenção da localização da entidade estática;

3.2. Na obtenção da hora do sistema;

3.3. Na obtenção do horário associado à entidade;

3.4. Na obtenção da localização principal;

3.5. Na obtenção da localização secundária;

3.6. Na obtenção da representação gráfica do objecto;

3.7. Na obtenção das restrições;

3.8. Na sobreposição com o mapa.

- Dependências:

Requisito Funcional	Depende de	Requisito Não-Funcional
RF 1	→	RNF 1.1
RF 2	→	RNF 1.2
RF 2.1	→	RNF 1.3
		RNF 2.1
		RNF 3.1
RF 2.2.1	→	RNF 1.4
		RNF 2.2
		RNF 3.2
RF 2.2.2	→	RNF 1.5
		RNF 2.3
		RNF 3.3
RF 2.2.3	→	RNF 1.6
RF 2.2.3.1	→	RNF 1.7

RF 2.2.3.1.2	→	RNF 1.8
		RNF 2.4
		RNF 3.4
RF 2.2.3.2	→	RNF 1.9
		RNF 2.5
		RNF 3.5
RF 3	→	RNF 1.10
		RNF 3.6
RF 4	→	RNF 1.11
		RNF 2.6
		RNF 3.7
RF 5	→	RNF 2.7
RF 6	→	RNF 1.12
		RNF 2.8
		RNF 3.8

- Actores:
  - Utilizador; Entidades; Sistema/Aplicação;

Modelação:

- **Estrutural:**
  - **Diagrama de *Features*:**

No diagrama abaixo – Figura 5.14 –, temos as características necessárias para apresentar uma entidade georreferenciada – *Show a Georeferenced Entity*. E existência de uma *Georeferenced Entity* é obrigatória e esta pode ser *Static Entity* ou *Mobile Entity*, não podendo ser as duas ao mesmo tempo. Tem de haver também, obrigatoriamente, uma *Location*, composta obrigatoriamente por uma *Longitude* e uma *Latitude* e podendo ter ou não uma *Altitude*. Opcionalmente podemos ter ainda um *Schedule*. Quando temos uma *Mobile Entity* é necessário que haja uma *Static Entity* e um *Schedule*. É ainda necessária a existência de um *Map* e de uma *Entity Graphical Representation*. E podemos ter opcionalmente alguma *Restriction/Information* relativa à entidade. Mais uma vez, a vantagem deste diagrama é identificar a variabilidade do padrão.

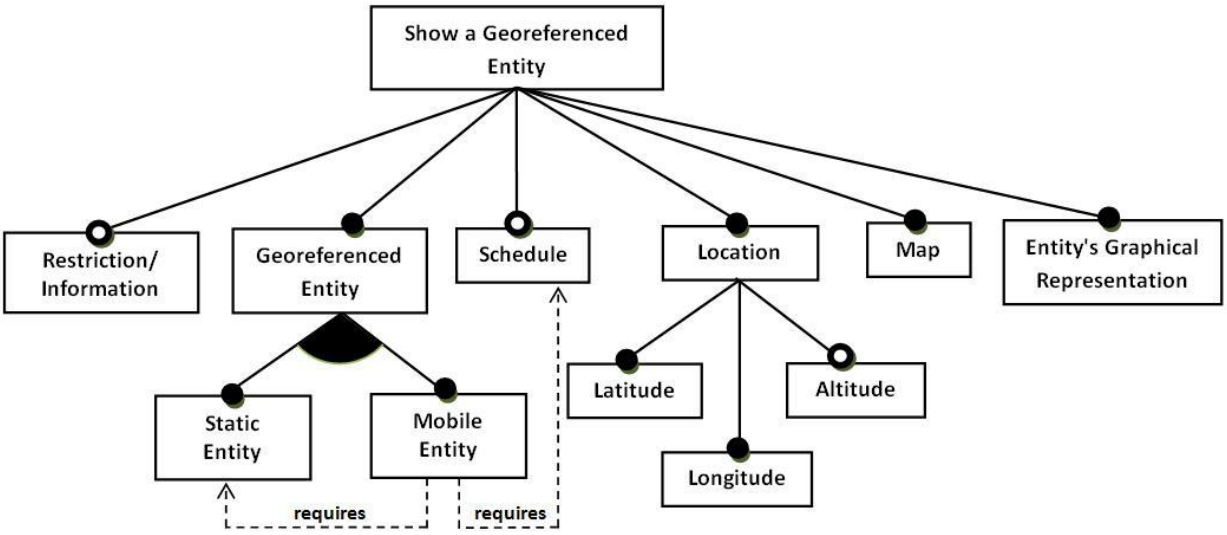

Figura 5.14. Diagrama de Features para Apresentar Entidade

- **Diagrama de Classes:**

No diagrama abaixo – Figura 5.15 –, temos 7 classes. A classe de *Interface*, a de *Control*, a *Locatable Entity* e temos ainda as classes *Schedule*, *Main Location*, *Secondary Location* e *Map*. A classe de *Interface* é a intermediária entre o utilizador e o *Control*, A classe de *Control* é a responsável por fazer as verificações, pedir os dados necessários e ainda comunica com as classes *Locatable Entity* e *Map*. A *Locatable Entity* tem um único *Schedule*, uma *Main Location*, pode ter várias *Secondary Location* e pode estar em vários *Map*. Um *Schedule* pertence apenas a uma *Locatable Entity*. Tanto uma *Main Location* como uma *Secondary Location* podem ser de várias *Locatable Entity*, bem como um *Map* pode ter várias *Locatable Entity* representadas.

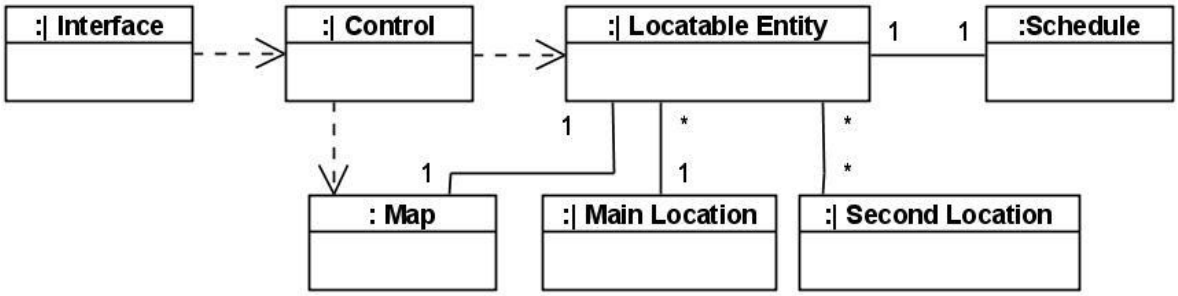

Figura 5.15. Diagrama de Classes para Apresentar Entidade

- **Comportamental – Diagrama de Sequência (Figura 5.16)**

O diagrama de Sequência apresentado na Figura 5.16 corresponde ao cenário aspectual do padrão. Este diagrama descreve o cenário onde é apresentada a representação gráfica de uma entidade georreferenciada (o aspecto). Como podemos ver, esta tem início no *Control* que

solicita a informação relativa à *Locatable Entity* e depois verifica se esta é estática ou móvel. Se a *Locatable Entity* for estática o *Control* informa a *Interface*, que obtém os dados de localização da mesma. Caso seja móvel, o *Control* também informa a *Interface* que obtém a hora do sistema e solicita, ao *Control*, o *Schedule* associado à *Locatable Entity*. O *Control* solicitada assim o mesmo à Entidade *Schedule* e verifica se, no momento, a *Locatable Entity* está na *Secondary Location* e informa a *Interface* do resultado. Se não estiver, a *Interface* solicita ao *Control* a verificação de se a *Locatable Entity* está em *working hour*. Se não estiver, a *Interface* é informada e envia uma mensagem ao Utilizador, a informar de que não é possível obter a localização da *Locatable Entity*. Se estiver em *working hour*, o *Control* informa a *Interface* e solicita à Entidade *Main Location* a sua localização. Se estiver na *Secondary Location*, o *Control* informa a *Interface* e solicita à entidade *Secondary Location* a sua localização, devolvendo-a depois à *Interface*. Feito isto, a *Interface* obtém a *Entity Graphical Representation* e depois questiona o *Control* relativamente à existência de alguma *Restriction/Information* a associar à *Locatable Entity*. Se esta existir, solicita ao *Control* a inserção dessa mesma na *Locatable Entity*. Seguidamente, o *Control* obtém o *Map*, envia para a *Interface* e esta, finalmente, sobrepõe a *Entity Graphical Representation* com o *Map*.

Consequências: A aplicação deste padrão irá permitir que se localize visualmente uma entidade de uma aplicação georreferenciada.

Lista de Eventos:

1. Ocorre um incêndio num edifício e é necessário apresentar todos os caminhos que possam ser tomados de forma a garantir que todos os ocupantes do mesmo saem do edifício;

2. Uma empresa de transportes por autocarro pretende indicar aos utilizadores onde está determinado autocarro em determinado momento, tendo de o apresentar visualmente num mapa;

3. Uma empresa que fornece serviços técnicos necessita de um sistema que lhe permita visualizar, num mapa, a localização de cada técnico a prestar assistência, de forma a poder gerir melhor o seu trabalho.

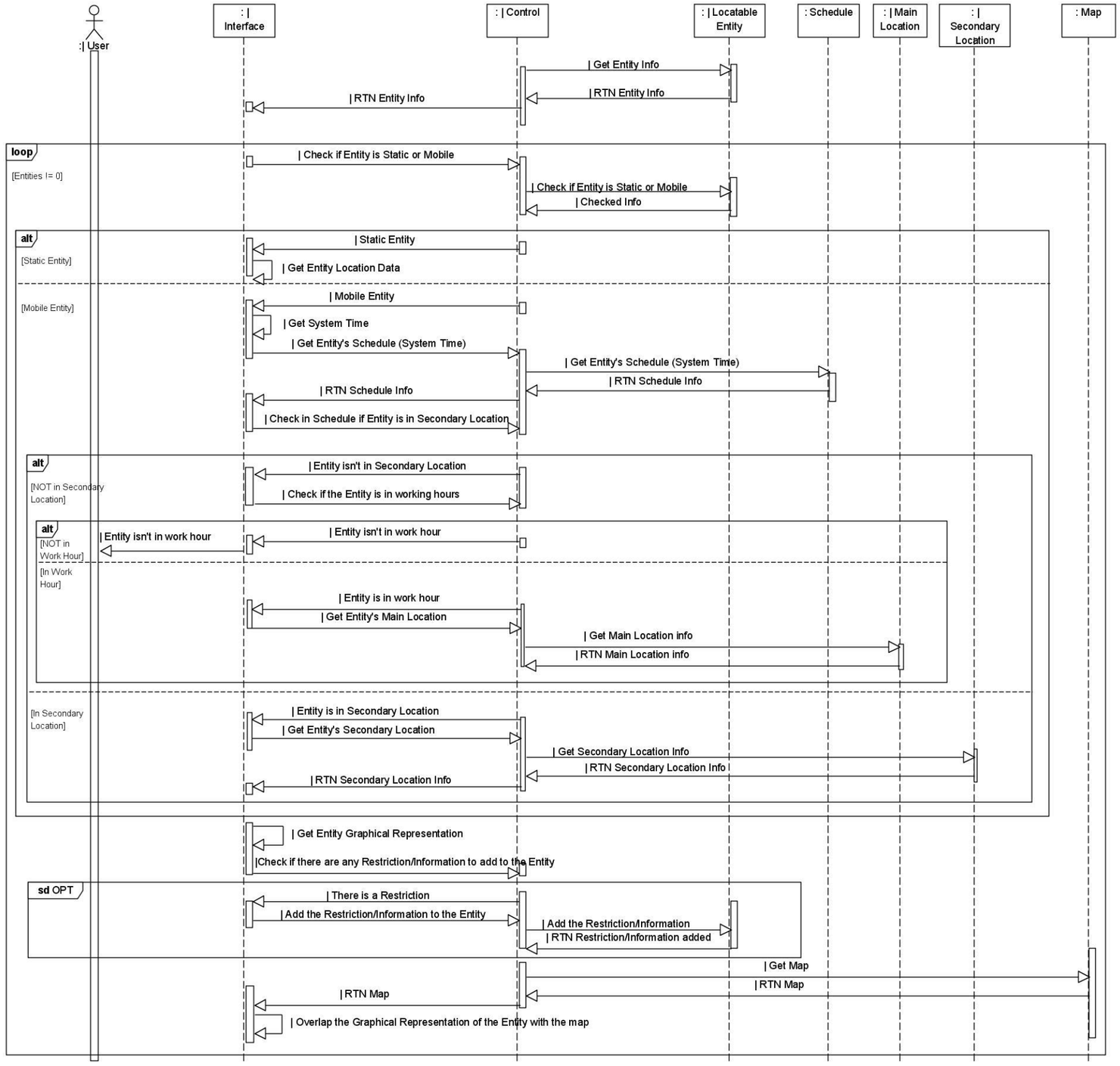

Figura 5.16. Diagrama de Sequência para Apresentar Entidade

Exemplos:

Este padrão pode ser aplicado a entidades estáticas, isto é, entidades que têm sempre a mesma localização e, neste caso, esta é simplesmente armazenada em Base de Dados. Por outro lado, o padrão pode também ser aplicado a entidades móveis, ou seja, entidades que mudam de localização de acordo com determinadas características, como é o caso de um indivíduo, ou de um veículo.

Como exemplo, vamos analisar a aplicação deste padrão ao sistema CLIP, o sistema de informação das disciplinas e horários de todos os alunos, docentes e restantes funcionários e colaboradores da Universidade Nova de Lisboa.

Vamos observar um exemplo de apresentação de uma entidade estática e de apresentação de uma entidade móvel.

Como exemplo de apresentação de entidade móvel, temos:

Acontece uma urgência e é necessário convocar uma reunião de professores. Para isso, é necessário saber onde os podemos encontrar, devendo para isso ser apresentada a localização destes num mapa.

Apresentar a Entidade Docente:

Modelação:

- **Diagrama de Features:**

No diagrama abaixo – Figura 5.17 –, temos as características necessárias para apresentar um docente – *Show Lecturer Entity*. Como se vê, é necessário ter um *Map*, um *Lecturer*, a *Lecturer's Graphical Representation*, o seu *Schedule* e a sua *Location*, sendo que esta última tem de ser dada por uma *Latitude* e uma *Longitude*, podendo também haver uma *Altitude*. Poderá existir também o *Office* do docente, uma *Room* e *Restriction/Information*.

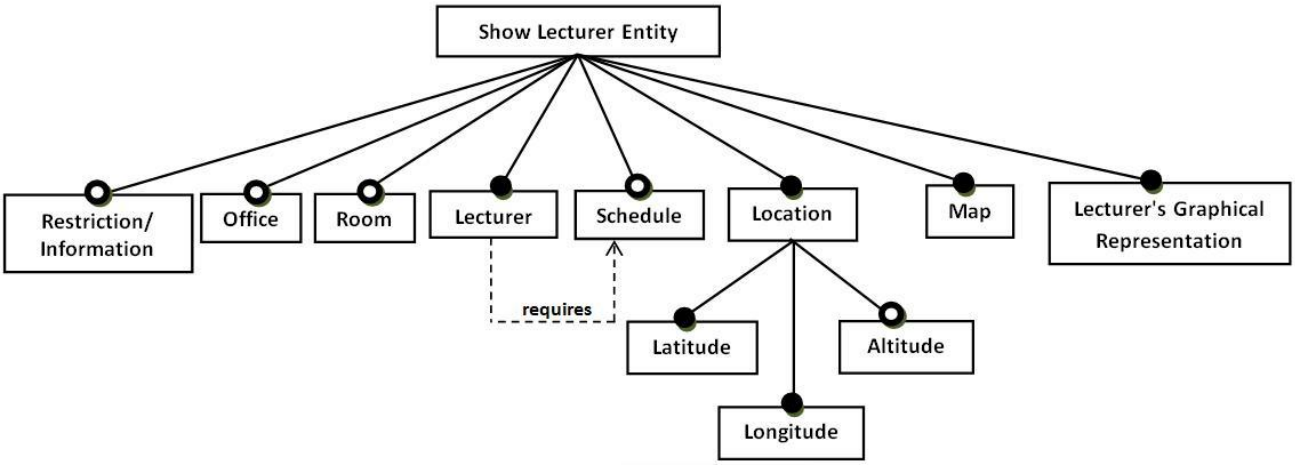

Figura 5.17. Diagrama de *Features* para Apresentar Docente

- **Diagrama de Classes:**

Na Figura 5.18 temos o diagrama de classes para o cenário base do exemplo, este cenário corresponde à apresentação num mapa dos docentes convocados para uma reunião. Assim, serão necessárias as classes *CLIP Interface*, *CLIP Control*, *Lecturer* e *Meeting*. Na Figura 5.19 temos o diagrama de classes para o cenário composto da base com o aspecto. Portanto, as classes do Aspecto são instanciadas com as do exemplo e acrescentamos as da base. Assim, as classes do Aspecto instanciadas são: *Interface* com *CLIP Interface*, *Control* com *CLIP Control*, *Locatable Entity* com *Lecturer*, *Main Location* com *Office* e *Secondary Location* com *Class Room*. As classes *Schedule* e *Map* não precisam de ser instanciadas e do cenário base acresce a classe *Meeting*.

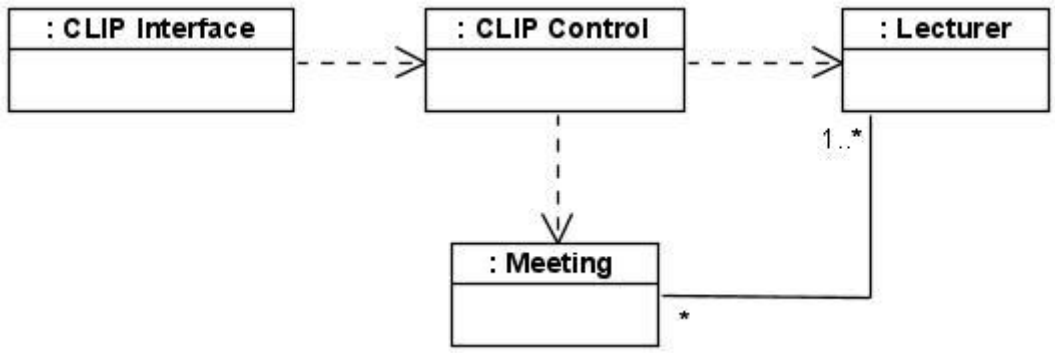

Figura 5.18. Diagrama de Classes do Cenário Base para Apresentar Docente

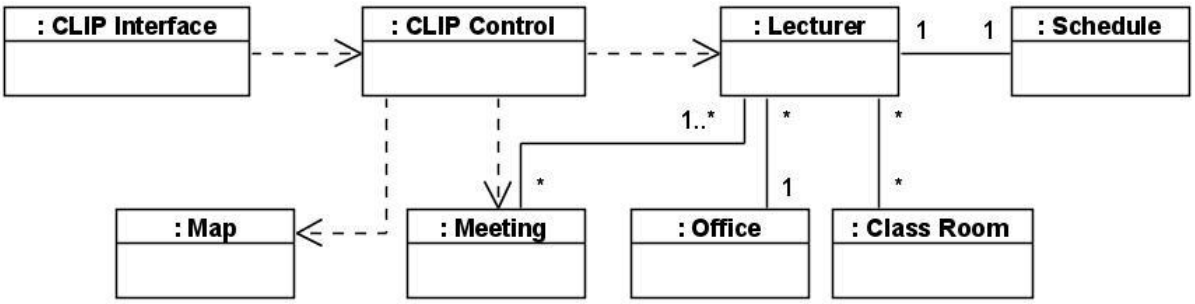

Figura 5.19. Diagrama de Classes do Cenário Composto para Apresentar Docente

- **Diagrama de Sequência:**

Na Figura 5.20 temos o diagrama de sequência para o cenário base do exemplo. Como já foi dito, este cenário corresponde à apresentação num mapa dos docentes convocados para uma reunião. É iniciado com o pedido de contacto dos docentes para uma reunião por parte do Utilizador. A *Interface* solicita ao *Control* a informação da *Meeting*, sendo que este faz o pedido da informação à entidade *Meeting*. A informação que a entidade *Meeting* devolve ao *Control* inclui a lista dos docentes que foram convocados para a mesma. Com esta informação o *Control* pede, à entidade *Lecturer,* a informação sobre cada docente e depois envia, para a *Interface,* a apresentação de cada docente, sendo depois a *Interface* a apresentar o docente ao Utilizador.

Na Figura 5.21, temos o diagrama de sequência para o cenário composto da base com o aspecto, e portanto temos a composição dos fluxos da base com os do aspecto. Assim, temos as entidades do Aspecto instanciadas com as do exemplo e acrescentamos as da base. As entidades do Aspecto instanciadas são: *Interface* com *CLIP Interface*, *Control* com *CLIP Control*, *Locatable Entity* com *Lecturer*, *Main Location* com *Office* e *Secondary Location* com *Class Room*. As classes *Map* e *Schedule* não precisam de ser instanciadas e do cenário base acresce a classe *Meeting*. Neste diagrama temos, no fundo, o cenário base até ao momento em que a informação do docente é devolvida ao controlo, sendo que depois disto temos o cenário aspectual – apresentado na Figura 5.16 – voltando-se depois, no final do cenário Aspectual, ao cenário base.

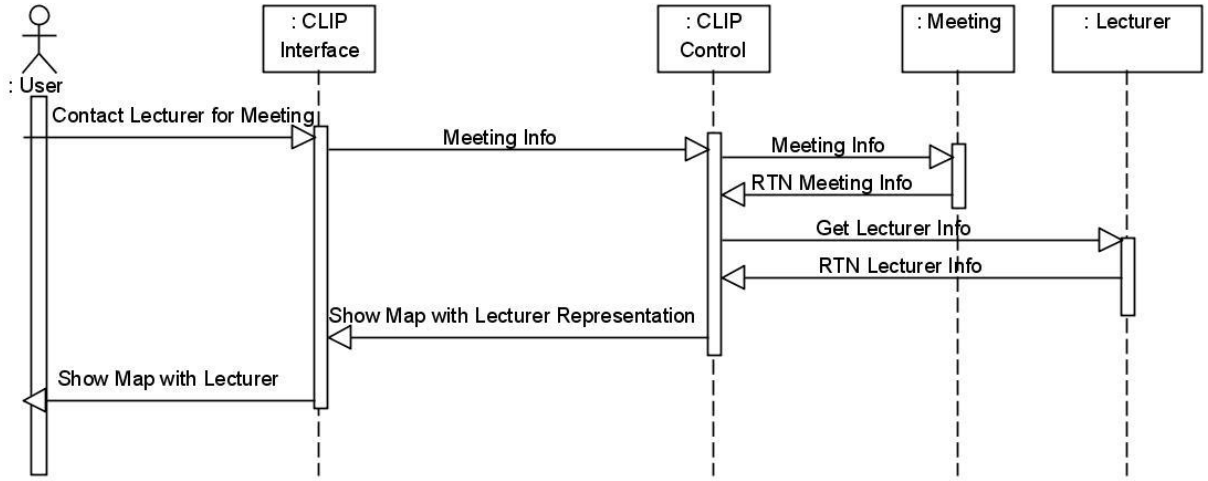

Figura 5.20. Diagrama de Sequência do Cenário Base para Apresentar Docente

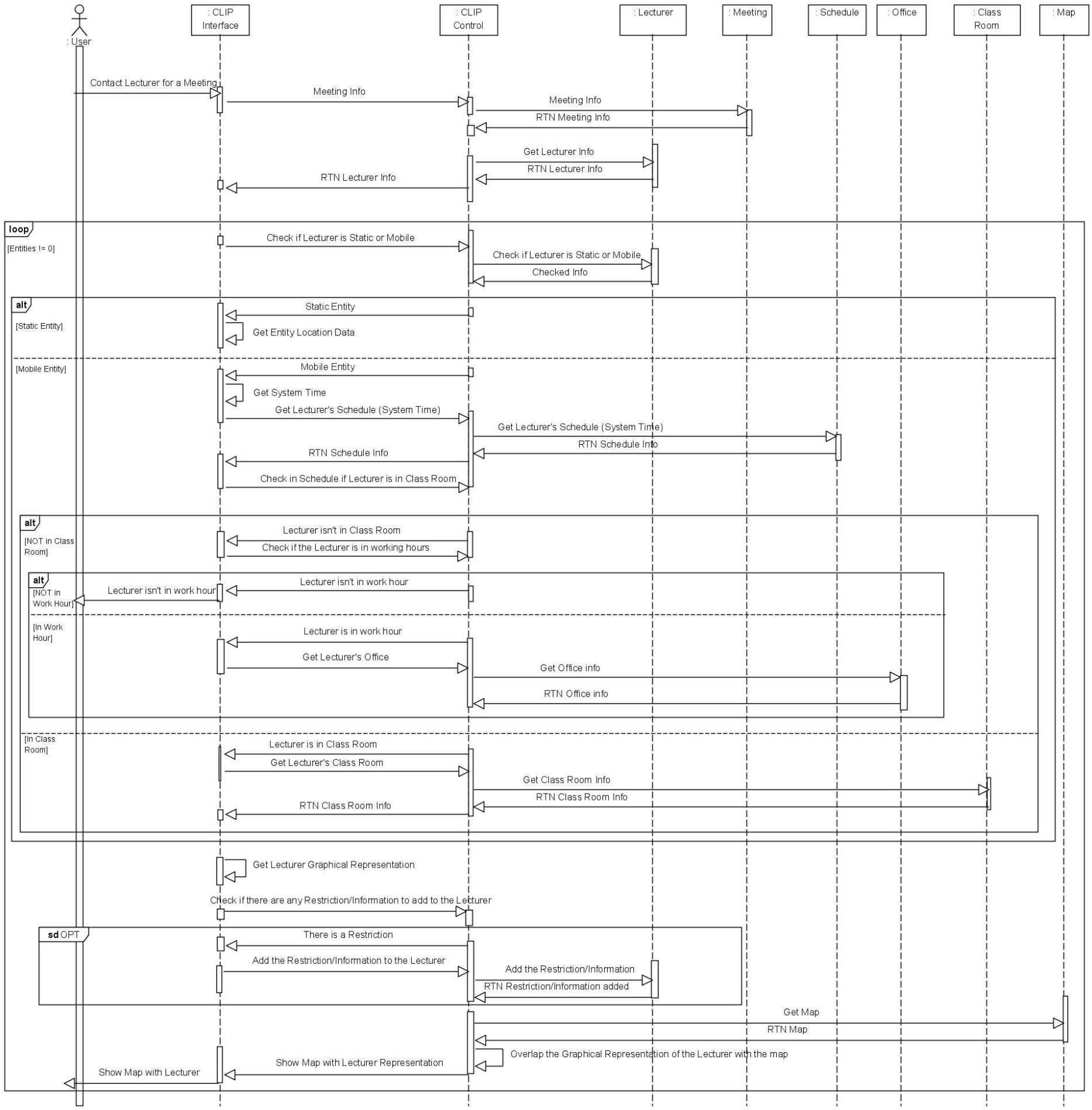

Figura 5.21. Diagrama de Sequência do Cenário Composto para Apresentar Docente

Como exemplo de apresentação de entidade estática, temos:

Um professor faz parte de um júri de uma dissertação de Mestrado, e precisa de saber onde a defesa da mesma, a que tem de assistir, vai decorrer. Assim, é necessário que este visualize num mapa a localização da sala em causa.

Apresentar a Entidade Sala de Aula:

Modelação:

- **Diagrama de Features:**

No diagrama abaixo – Figura 5.22 –, temos as características necessárias para apresentar o local onde vai decorrer uma discussão de tese – *Show Class Room Entity*. Como se vê, é necessário ter um *Map*, uma *Class Room*, a sua *Graphical Representation* e a sua *Location*, sendo que esta última tem de ser dada por uma *Latitude* e uma *Longitude*, podendo também haver uma *Altitude*. Poderão haver também *Restriction/Information* a adicionar à *Class Room*.

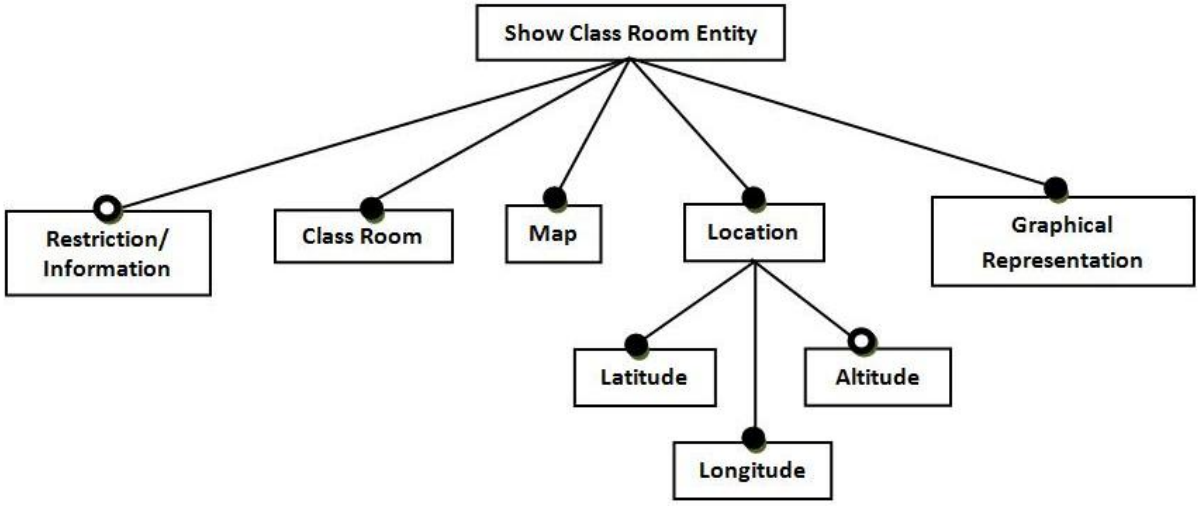

Figura 5.22. Diagrama de *Features* para Apresentação da Sala de Aula

- **Diagrama de Classes:**

Na Figura 5.23 temos o diagrama de classes para o cenário base do exemplo, este cenário corresponde à apresentação num mapa da localização da Sala de Aula onde vai decorrer a sessão de defesa de uma dissertação. Assim, serão necessárias as classes *CLIP Interface*, *CLIP Control* e *Thesis Discussion*. Na Figura 5.24, temos o diagrama de classes para o cenário composto da base com o aspecto. Desta forma, as classes do Aspecto são instanciadas com as do exemplo e acrescentamos as da base. Assim as classes do Aspecto instanciadas são: *Interface* com *CLIP Interface*; *Control* com *CLIP Control*; *Thesis Discussion*.

Figura 5.23. Diagrama de Classes do Cenário Base para Apresentar a Sala de Aula

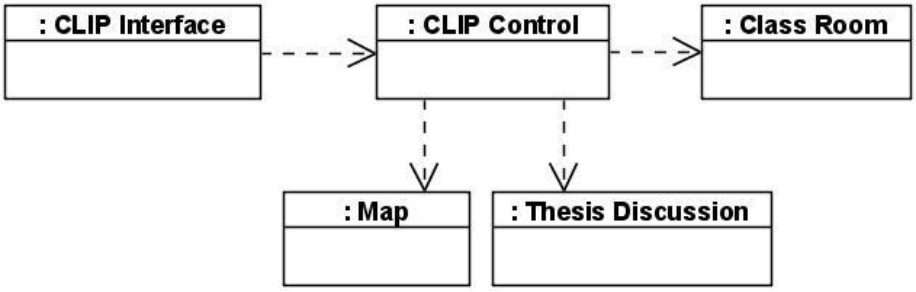

Figura 5.24. Diagrama de Classes do Cenário Composto para Apresentar a Sala de Aula

- **Diagrama de Sequência:**

Na Figura 5.25 temos o diagrama de sequência para o cenário base do exemplo. Como já foi dito, este cenário corresponde à apresentação num mapa de uma Sala de aula onde vai decorrer a sessão de defesa de uma dissertação. É iniciado com o pedido de apresentação do local onde vai decorrer a sessão, por parte do Utilizador. A *Interface* solicita ao *Control* a informação da *Thesis Discussion*, sendo que este faz o pedido da informação à entidade *Thesis Discussion*. A informação que esta devolve ao *Control* inclui a identificação da *Class Room* onde vai decorrer a discussão. Feito isto, o *Control* envia para a *Interface* a apresentação do mapa com a representação da *Class Room*, sendo depois esta a mostrar a mesma ao Utilizador.

Na Figura 5.26, temos o diagrama de sequência para o cenário composto da base com o aspecto, e portanto temos a composição dos fluxos da base com os do aspecto. Desta forma, temos as entidades do Aspecto instanciadas com as do exemplo e acrescentamos as da base. Assim, as entidades do Aspecto instanciadas são: *Interface* com *CLIP Interface*; *Control* com *CLIP Control*; e *Locatable Entity* com *Class Room*. A *Main Location* e a *Secondary Location* não são instanciadas porque, como já foi dito, *Class Room* é uma entidade estática. As classes *Schedule* e *Map* não precisam de ser instanciadas e do cenário base acresce a classe *Thesis Discussion*. Neste diagrama temos, no fundo, o cenário base até ao momento em que a informação da sessão é devolvida ao *CLIP Control*, sendo que depois disto temos o cenário

aspectual – apresentado na Figura 5.16 – voltando-se depois, no final do cenário Aspectual, ao cenário base.

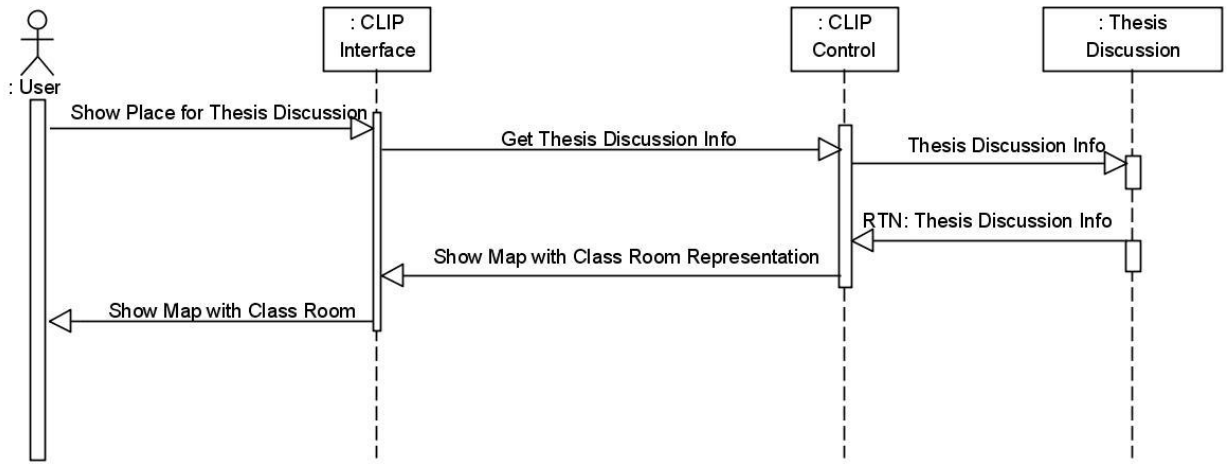

Figura 5.25. Diagrama de Sequência do Cenário Base para Apresentar a Sala de Aula

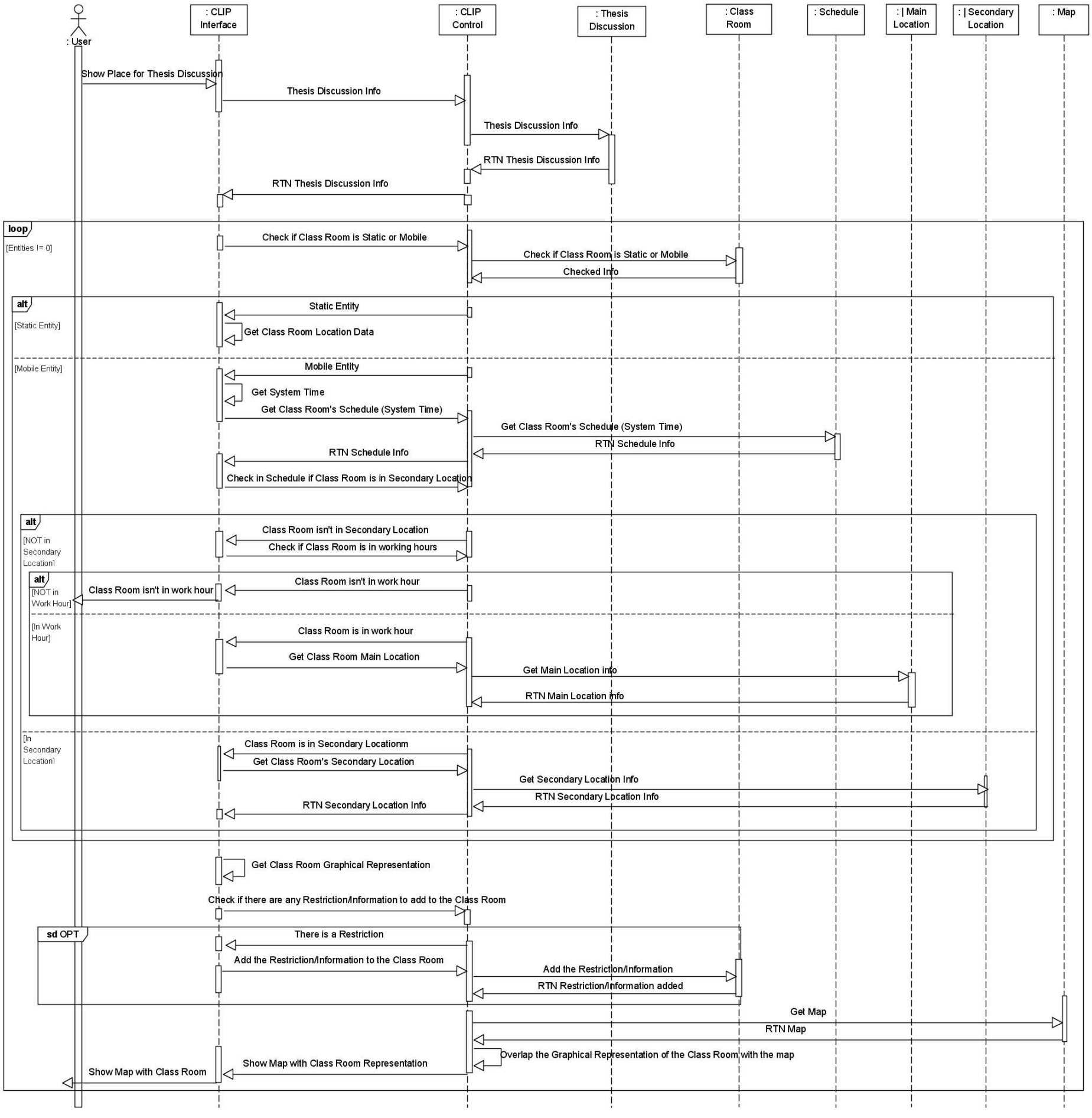

Figura 5.26. Diagrama de Sequência do Cenário Composto para Apresentar a Sala de Aula

Padrões Relacionados (Urbieta, 2010):

- Localizar Espacialmente uma Entidade de uma Aplicação sem Georreferenciação;
- Adicionar disponibilidade temporal às entidades espaciais;
- Ajustar o estado das entidades (Adicionar/Remover todas ou parte das entidades espaciais).

5.2. Sumário

Ao longo deste capítulo explicámos o *template* de padrão de análise usado nesta dissertação para descrever os padrões. Explicou-se como foram obtidos os padrões de análise que são descritos nesta dissertação e quais os restantes padrões que foram identificados. Estes padrões foram identificados através de uma análise mais detalhada das diferentes classes da *API* do *Google Maps*, em conjunto com os *standards* existentes de SIG[5], de forma a identificar padrões através da detecção de chamadas à *API* que ocorrem diversas vezes e que podem ser disponibilizadas como funções.

Foi feita também uma breve apresentação do processo de descrição dos padrões.

Finalmente, apresentou-se a descrição dos dois padrões seleccionados para serem descritos e apresentou-se uma justificação para a escolha destes dois em detrimento dos restantes.

[5] www.opengis.org

6. *PatternTool*

Ao longo deste capítulo vai ser apresentada a ferramenta *PatternTool*. Esta ferramenta foi concebida com vista a suportar computacionalmente a usabilidade da abordagem desenvolvida nesta dissertação, para facilitar e promover a reutilização dos padrões. Assim, funciona como suporte para a criação de *templates* de padrões para que estes possam ser integrados num catálogo de padrões.

Para conceber esta ferramenta foi criada uma Linguagem de Domínio Especifico (LDE). Uma LDE (Hudak, 1998), do inglês *Domain Specific Language* (DSL), é uma linguagem de programação adaptada a um problema específico. Isto é, oferece, através das devidas notações e abstracções, uma boa expressividade e foca-se num problema de um determinado domínio, sendo assim capaz de captar, de forma precisa e exacta, a semântica de um domínio de aplicação. Desta forma, uma LDE pode ser usada para gerar partes de um conjunto de sistemas num domínio de aplicação. Quando uma LDE é baseada num conhecimento profundo do domínio da aplicação, fornece a expressividade necessária para que se gerem os componentes necessários de forma simples.

Batory (2002) defende que as LDEs, no seu domínio de aplicação, oferecem um ganho substancial na expressividade e na usabilidade, quando comparadas com as Linguagens de Programação de propósito Geral (LPG), do inglês *General-purpose Programming Languages* (GPL). Dado que para o desenvolvimento das LDEs é necessário que haja um conhecimento e experiência profundo do domínio e no desenvolvimento da linguagem, e poucas pessoas têm experiência em ambos, o desenvolvimento das mesmas torna-se pesado, ficando muitas vezes, o seu desenvolvimento para segundo plano.

Segundo Hudak (1998), uma LDE permite que se desenvolva *software* para um determinado domínio de aplicação de forma rápida e eficaz, originando programas fáceis de compreender e de manter. Dursen *et* al. (2002) acrescentaram ainda, a grande portabilidade, confiança, optimização e testabilidade, como vantagens das LDEs.

A principal vantagem das LDEs prende-se com o facto de estas oferecerem um alto nível de especificidade de domínio das melhores formas possíveis. Isto é, desde o inicio da sua concepção, as LDEs oferecem notações dedicadas ao domínio específico. Estas notações aumentam a legibilidade e tornam a especificação acessível aos utilizadores do domínio, não tendo estes de ser programadores. Estas notações estão directamente relacionadas com o aumento da produtividade associado ao uso das LDEs (Mernik *et* al., 2005 e Thibault, 1998).

O método de construção das LDEs permite que haja reutilização de desenho, pois estas facilitam o delineamento do conjunto de decisões de desenho que devem ser tomadas para obter um membro do programa (Thibault, 1998).

Sendo as LDEs linguagens de alto nível, beneficiam das características deste tipo de linguagens. Isto é, têm a capacidade de reduzir a propensão a erros e de abstrair dos detalhes da implementação de baixo nível, produzindo especificações mais concisas e levando à redução do tempo de colocação no mercado de novos produtos (Thibault, 1998).

Assim, as LDEs têm a grande vantagem de produzir programas fáceis de escrever, de interpretar e de modificar, quando comparadas com os produzidos pelas LPGs (Dursen *et* al., 2002).

6.1. Sintaxe e Semântica da Linguagem

De acordo com Harel *et* al. (2000), e como podemos ver na Figura 6.1, a linguagem é composta por uma componente de noções sintácticas (sintaxe) que pode ser um conjunto de elementos que são usados para comunicar em conjunto com o seu significado (semântica).

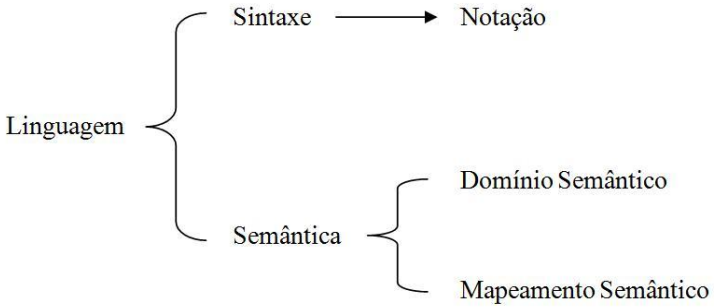

Figura 6.1. Estrutura da Linguagem

A sintáctica consiste na notação da linguagem, abstraindo-se completamente do significado. O significado da linguagem é descrito pela semântica. A semântica da linguagem dá-nos o

significado de cada uma das suas expressões. Esse significado deve ser um elemento de um domínio bem definido.

A semântica tem duas componentes: o domínio semântico e o mapeamento semântico da sintaxe para o domínio semântico.

O domínio semântico especifica todos os conceitos existentes no universo do domínio, sendo uma abstracção da realidade que descreve os aspectos importantes dos sistemas que tencionamos desenvolver. O domínio semântico é definido com o propósito de descrever o significado de uma notação, sendo que a definição do domínio semântico é, normalmente, independente da notação. Isto permite a reutilização do domínio semântico para outras notações.

6.2. Desenvolvimento Orientado a Modelos

Segundo Ozgur (2007), ao contrário das LPGs, as LDEs trazem a modelação para muito mais próximo dos *experts* do domínio e tornam mais fácil a manutenção e a evolução dos modelos que contribuem para o aumento da produtividade e para a agilidade do Desenvolvimento Orientado a Modelos, do inglês *Model Driven Development* (MDD).

Um modelo é uma descrição abstracta do *software* que esconde informação sobre alguns aspectos do *software*.

Actualmente, acredita-se que os modelos podem ser bastante benéficos nas fases iniciais do desenvolvimento para esboçar e comunicar o significado das ideias de desenho de alto nível. Assim, os modelos passam a ter um papel essencial no MDD. O MDD aplica modelos para representar os elementos de um sistema e as suas relações. Os modelos servem como *input* e *output* para todas as fases de desenvolvimento de um sistema, até este ser gerado. Nesta dissertação os modelos têm um papel fundamental nas descrições dos padrões.

Existem duas principais preocupações no *core* do MDD:

- *Elevar o nível de abstracção das especificações* para se aproximar do domínio do problema, usando linguagens de modelação de nível mais elevado e construções com melhor comportamento.
- *Elevar o nível de automatismo*, usando tecnologias para colmatar a lacuna semântica existente entre a especificação (modelo) e a implementação (código gerado). Desta forma,

o MDD pode fornecer um automatismo extensivo de algumas actividades usando informação capturada pelos modelos.

Os efeitos destas preocupações traduzem-se numa maior qualidade do produto e no aumento da produtividade no desenvolvimento.

Os modelos são descritos por metamodelos. Um metamodelo é uma especificação explícita de uma abstracção, onde se identifica a lista de conceitos e a lista de relações relevantes entre estes conceitos.

A metamodelação identifica conceitos gerais e as suas relações num determinado problema de domínio e origina uma linguagem de modelação usada para criar modelos de domínio executáveis. O modelo executável deve representar os conceitos do domínio destino onde o sistema de *software* irá ser usado. Um metamodelo especifica o âmbito do que deve ser definido num modelo, que elementos devem ser incluídos num modelo e como estes elementos se relacionam uns com os outros. O metamodelo abstrai das especificações das tecnologias de implementação, focando-se na definição de conceitos do domínio.

A metamodelação é usada para criar a sintaxe abstracta de uma linguagem de modelação. O *Meta Object Facility* (MOF) é uma técnica de metamodelação *standardizada* pela OMG onde o metamodelo é escrito como uma simplificação do diagrama de classes UML e a linguagem de restrições a objectos, do inglês *Object Constraint Language* (OCL), é usada para definir restrições na sintaxe abstracta.

Metamodelos, modelos e instâncias compreendem diferentes níveis de abstracção que se podem ver representados na Figura 6.2:

- M3 – Nível do Meta-Metamodelo: contém apenas o MOF;
- M2 – Nível do Metamodelo: Contém qualquer tipo de metamodelo, incluindo o metamodelo UML.
- M1 – Nível do Modelo: Contém qualquer modelo com um metamodelo correspondente em M2;
- M0 – Nível Concreto: Qualquer situação real, única em espaço e tempo, representada por um determinado modelo em M1.

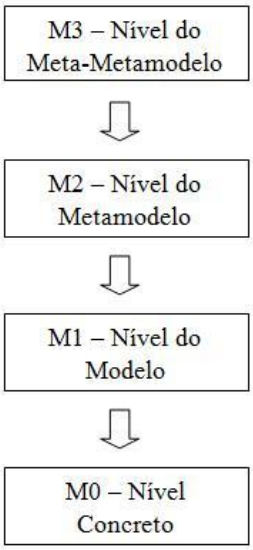

Figura 6.2. Metamodelação

A metamodelação facilita muito a implementação de uma LDE e oferece suporte para a experimentação com linguagem de modelação tal como ela é construída. Desta forma, a definição da linguagem baseada em metamodelo também ajuda na tarefa de construção de geradores para reduzir o peso da criação e manutenção da ferramenta (Ozgur, 2007)

6.3. *Framework* e *plugins* para Implementação da ferramenta *PatternTool*

Para implementar a ferramenta de construção de padrões geoespaciais, foi usado o ambiente Eclipse Galileo[6] com a *framework* EMF[7]/GMF[8] e os *plugins* Emfatic (Emfatic, 2008) e o EuGENia (EuGENia, 2008).

O Eclipse é uma plataforma de modelação que suporta a metamodelação e além disso é altamente flexível e *open source*. O *Eclipse Modelling Framework* (EMF) pode gerar código fonte a partir de modelos definidos através da definição do diagrama de classes UML, mas este não contém a definição da sintaxe concreta. O *Graphical Editing Framework* (GEF) é também um projecto Eclipse, e oferece métodos para criar editores visuais. O EMF não suporta geração de código para o GEF, portanto este necessita de codificação manual para suportar a sintaxe concreta.

[6] Eclipse Galileo, http://www.eclipse.org/downloads/packages/release/galileo/sr2, 2009.

[7] *Eclipse Modelling Framework*, http://www.eclipse.org/modeling/emf/?project=emf, 2004.

[8] *Graphical Modeling Framework*, http://www.eclipse.org/modeling/gmf/, 2004.

O *Graphical Modeling Framework* (GMF) também é um projecto Eclipse. O seu objectivo é formar uma ligação generativa entre o EMF e o GEF, para que a definição do diagrama seja ligada ao modelo do domínio como um *input* para a geração de um editor visual. O GMF usa uma LDE de apresentação para definir a sintaxe concreta. O resultado (as definições de ligações concretas e estruturais) é processado essencialmente para produzir código fonte. O mapeamento entre o modelo do domínio e os itens do modelo da sintaxe concreta também são suportados no GMF. O código fonte gerado depende das características do GEF e EMF. Embora o conceito do GMF seja simples, tem alguns pontos fracos:

- A geração não é baseada na transformação do modelo. Consequentemente, os passos da compilação são codificados manualmente, assim, mudar a transformação implica mudar o código fonte e reconstruir o compilador. No caso de transformação de modelos tais transformações podem ser realizadas em tempo de execução.
- Devido ao EMF, o GMF é restrito apenas para Java. (Mezei *et* al., 2006)

O Emfatic é um editor de texto que suporta navegação, edição e conversão do modelo Ecore, usando uma sintaxe compacta e perceptível, semelhante ao Java. A versão do Emfatic lançada em 2005 suporta o EMF genérico, sistema de ficheiros, hierarquia EMF, *hyperlinks*, auto-edição, e a possibilidade de definir *templates* para acelerar a criação de documentos, além de outras características de usabilidade. A melhor forma de experimentar o Emfatic é clicar com o botão direito num ficheiro *.ecore* e escolher *Generate Emfatic source*, esta acção vai correr um conversor que vai produzir um ficheiro Emfatic com o metamodelo gerado.

O EuGENia é uma ferramenta que gera automaticamente os modelos *.gmfgraph*, *.gmftool* e *.gmfmap* necessários para implementar um editor GMF a partir de um único metamodelo Ecore anotado. Por exemplo, a partir de um metamodelo EMF gerado através do Emfatic, adicionam-se as anotações do EuGENia e podemos obter um editor GMF completamente funcional. Na Figura 6.3, apresenta-se o exemplo de um pedaço do metamodelo *emf* do editor principal produzido para a ferramenta criada nesta dissertação. Este metamodelo *emf* foi produzido através do Emfatic com algumas anotações do EuGENia:

```
@namespace(uri="padrao", prefix="padrao")
package padrao;

@gmf.diagram(foo="bar")
class Padrao {
  val PadraoTemplate[1] Padrao;
}
```

```
@gmf.node(figure="rectangle", label="nomePadrao", label.pattern="Pattern Template", tool.name="Pattern Template")
class PadraoTemplate {
 attr String nomePadrao;

 @gmf.compartment(layout="list")
 val NomePadrao[1] nomePad;

 @gmf.compartment(layout="list")
 val ReqsPadrao[1] PadReqs;
}

@gmf.node(figure="rectangle", label="nomeP", label.pattern="Name: {0}", tool.name="Pattern's Name", tool.description="Add the Pattern's Name")
class NomePadrao {
 attr String nomeP;
}

@gmf.node(figure="rectangle", label="nomeR", label.pattern="Requirements: {0}", tool.name="Pattern's Requirements Analysis", tool.description="Add Pattern's Requirements Analysis")
class ReqsPadrao {
 attr String nomeR;

 val Requisitos[2] ReqPad;

 @gmf.compartment(layout="list")
 val Funcionais[+] f;

 @gmf.compartment(layout="list")
 val NaoFuncionais[+] nf;

 @gmf.compartment(layout="list")
 val Dependencias[1] DepsPad;
}

@gmf.link(source="LigacaoIn", target="LigacaoOut", style="dash", width="2", color="255,69,0", target.decoration="arrow", tool.name="Dependency Connection Between Requirements", tool.description="Add a Dependency Connection Between Requirements")
class Ligacao {
 ref Requisitos LigacaoIn;
 ref Requisitos[1] LigacaoOut;
}

@gmf.node(label="ident, descricao")
abstract class Requisitos {
 attr String ident = "1";
 attr String descricao;
 val Ligacao[*] ligacoes;
}
@gmf.node(figure="rounded", label.pattern="FR{0}. {1}", label.icon="false", tool.name="Functional
```

```
Requirement", tool.description="Add a Functional Requirement to the Requirements Section")
class Funcionais extends Requisitos {
}

@gmf.node(figure="ellipse", label.pattern="NFR{0}. {1}", label.icon="false", tool.name="Non-Functional Requirement", tool.description="Add a Non-Functional Requirement to the Requirements Section")
class NaoFuncionais extends Requisitos {
}
```

Figura 6.3. Metamodelo *emf*

Como se pode ver no exemplo da Figura 6.3, o EuGENia suporta diversas anotações nos elementos do metamodelo. Abaixo apresentamos algumas das principais:

@gmf.diagram – Indica o objecto raiz do metamodelo. Só pode haver uma metaclasse, em todo o metamodelo, com esta anotação. Esta permite especificar alguns parâmetros relativamente ao modelo do domínio e ao diagrama;

@gmf.node – Quando aplicado a uma metaclasse indica que esta deve aparecer no diagrama como um nó. Para esta anotação também é possível especificar diversos parâmetros, como características da *label*, da figura associada ao nó, cores, tamanhos e a forma como o nó é especificado na palete;

@gmf.link – Esta anotação pode ser aplicada a metaclasses que devem aparecer no diagrama como *links* e a ligações. De acordo com o tipo de elemento a que são aplicados existem parâmetros diferentes que podem ser especificados na anotação. Mas de um modo geral podem fazer-se especificações sobre a origem e destino do *link*, sobre a *label*, o tipo de seta usada na origem e no destino do *link*, a cor, a espessura e o tipo de linha usado no *link* e a forma como ele surge na palete.

@gmf.compartment – Define que é definido um compartimento onde serão incluídos os elementos do tipo especificado. Esta anotação permite, também alguns parâmetros relativamente ao *layout* do compartimento.

6.4. Criação do Modelo Ecore para a Ferramenta *PatternTool*

Para a criação da LDE foi necessário criar um metamodelo, para isto foi usado o modelo Ecore. Neste modelo especificaram-se todas as regras que se pretendiam ter na LDE. O modelo Ecore para a linguagem *PatternTool* é apresentado abaixo, na Figura 6.4.

Como podemos ver nesta figura, o modelo tem um nó raiz chamado *Padrão*[9], que contém um *template* de padrão, o nó *PadraoTemplate*. A este nó estão ligadas as metaclasses que representam os campos do *template*. Estes são as metaclasses *NomePadrao*, *ProblemaPadrao*, *ContextoPadrao*, *ReqsPadrao*, *ListaEventosPadrao*, *Modelacao*, *Consequencias*, *Exemplos* e *PadroesRelacionados*. Sendo que alguns campos têm sub-campos, como é o caso do *ReqsPadrao*, *Modelacao* e *Exemplos*. No caso do campo *ReqsPadrao*, este é composto por Requisitos Funcionais – metaclasse *Funcionais* –, Requisitos Não-Funcionais – metaclasse *NaoFuncionais* –, Dependências entre requisitos – metaclasse *Dependencias* – e por Actores – metaclasse *Actores*. No caso do campo *Modelacao*, este é composto por 2 tipos de modelação, a Modelação Estrutural – metaclasse *Estrutural* – e a Modelação Comportamental – metaclasse *Comportamental*. A Modelação Estrutural é efectuada através dos diagramas de Features – metaclasse *DiagFeatures* – e de Classes – metaclasse *DiagClasses* – e a Modelação Comportamental é obtida através do Diagrama de Sequência – metaclasse *DiagSequencia*. Estes três diagramas fazem parte do *template*, mas a sua edição é feita em sub-editores que foram criados para o efeito. No caso do campo *Exemplos*, este é composto por um bloco de Requisitos Funcionais, Diagramas de *Features*, Diagramas de Classes e Diagramas de Sequência para permitir a utilização de cenários.

[9] Os metamodelos da Linguagem, ao contrário do resto da ferramenta, estão em Português porque inicialmente começou a criar-se tudo neste idioma, no entanto, mais tarde sentiu-se a necessidade de ter a ferramenta numa linguagem que fosse perceptível por outras nacionalidades, tendo-se, desta forma alterado o idioma da ferramenta e dos diagramas contidos na descrição dos padrões para Inglês.

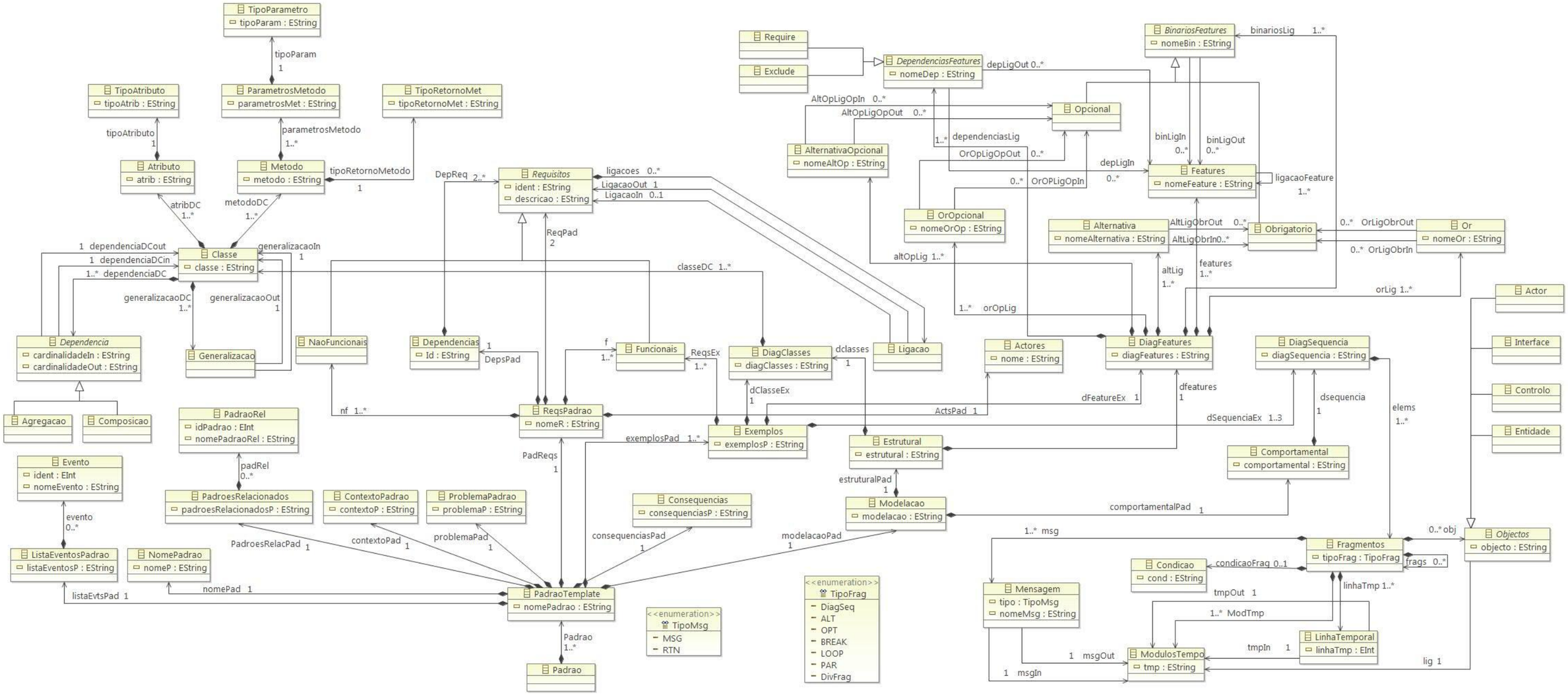

Figura 6.4. Modelo Ecore da Linguagem *PatternTool*

De seguida vão explicar-se os conceitos fundamentais do metamodelo e qual o seu papel na linguagem. Vai dividir-se esta explicação por editores, começando-se pelos conceitos do metamodelo que integram o editor principal, do *Template* do Padrão, seguindo-se os conceitos de cada um dos sub-editores dos diagramas. Para cada editor vamos ter o metamodelo Ecore apenas com as metaclasses que integram o editor em causa e uma tabela onde são explicados os conceitos do mesmo.

Abaixo apresenta-se a Tabela 6.1, onde são explicados os conceitos presentes no metamodelo Ecore da Figura 6.5, sendo que a tabela e o metamodelo correspondem ao editor principal.

Tabela 6.1. Editor do *Template* de Padrão

Elemento	**Descrição**
PadraoTemplate	Representa o *template* de descrição do padrão. Assim, tem agregados a si todos os campos que compõe a descrição do padrão.
NomePadrao	Representa o nome que é dado ao padrão, que servirá para o identificar.
ProblemaPadrao	Representa o tipo de problemas que podem beneficiar da aplicação do padrão.
ContextoPadrao	Representa o contexto em que o padrão se poderá aplicar.
ListaEventosPadrao	Consiste num compartimento onde vão estar listados os eventos que poderão despoletar este padrão.
Evento	Representa um evento que poderá activar o padrão.
ReqsPadrao	Representa o compartimento da descrição dos requisitos do padrão. Assim, tem agregados a si os requisitos funcionais, os requisitos não-funcionais, as dependências existentes entre os requisitos e finalmente os actores que intervêm neste padrão.
Funcionais	Representa os requisitos funcionais do padrão. Podem existir diversos requisitos funcionais num padrão.
NaoFuncionais	Representa os requisitos não-funcionais do padrão. Podem existir diversos requisitos não-funcionais num padrão.
Dependencias	Representa o compartimento onde se vão colocar as dependências entre os requisitos funcionais e não-funcionais.
Ligacao	Consiste numa ligação entre requisitos e representa as dependências existentes entre os requisitos que estão ligados por ela.
Actores	Representa as entidades intervenientes no padrão.
Modelacao	Representa o compartimento de modelação do padrão. Tem a si agregados os compartimentos de Modelação Estrutural e de Modelação Comportamental.
Estrutural	Representa o compartimento de Modelação Estrutural do padrão. Tem agregados a si os diagramas de *Features* e de Classes.
DiagFeatures	Este elemento representa um diagrama de *Features*. Através deste tem-se acesso a um sub-editor onde se pode construir e editar o diagrama de *Features*. É ainda neste mesmo

	compartimento que irá aparecer o diagrama depois de salvo no sub-editor.
DiagClasses	Este elemento representa um diagrama de Classes. Através deste tem-se acesso a um sub-editor onde se pode construir e editar o diagrama de Classes. É ainda neste mesmo compartimento que irá aparecer o diagrama depois de salvo no sub-editor.
Comportamental	Representa o compartimento de Modelação Comportamental do padrão. Tem agregado a si o diagrama de Sequência.
DiagSequencia	Este elemento representa um diagrama de Sequência. Através deste tem-se acesso a um sub-editor onde se pode construir e editar o diagrama de Sequência. É ainda neste mesmo compartimento que irá aparecer o diagrama depois de salvo no sub-editor.
Consequencias	Representa os efeitos que se fazem sentir com a aplicação do padrão, podem ser consequências positivas ou negativas.
Exemplos	Representa o compartimento onde são apresentados exemplos de aplicação do padrão. A descrição dos exemplos é feita por meio de requisitos funcionais, diagrama de *Features*, de um diagramas de Classes e diagramas de Sequência. Desta forma, terá agregados a si, um compartimento de requisitos funcionais, um de diagrama de *Features*, um de diagrama de Classes e um de diagrama de Sequência.
PadroesRelacionados	Consiste num compartimento onde vão estar listados os padrões relacionados com o padrão que está a ser descrito.
PadraoRel	Representa um padrão relacionado com o padrão que está a ser descrito.

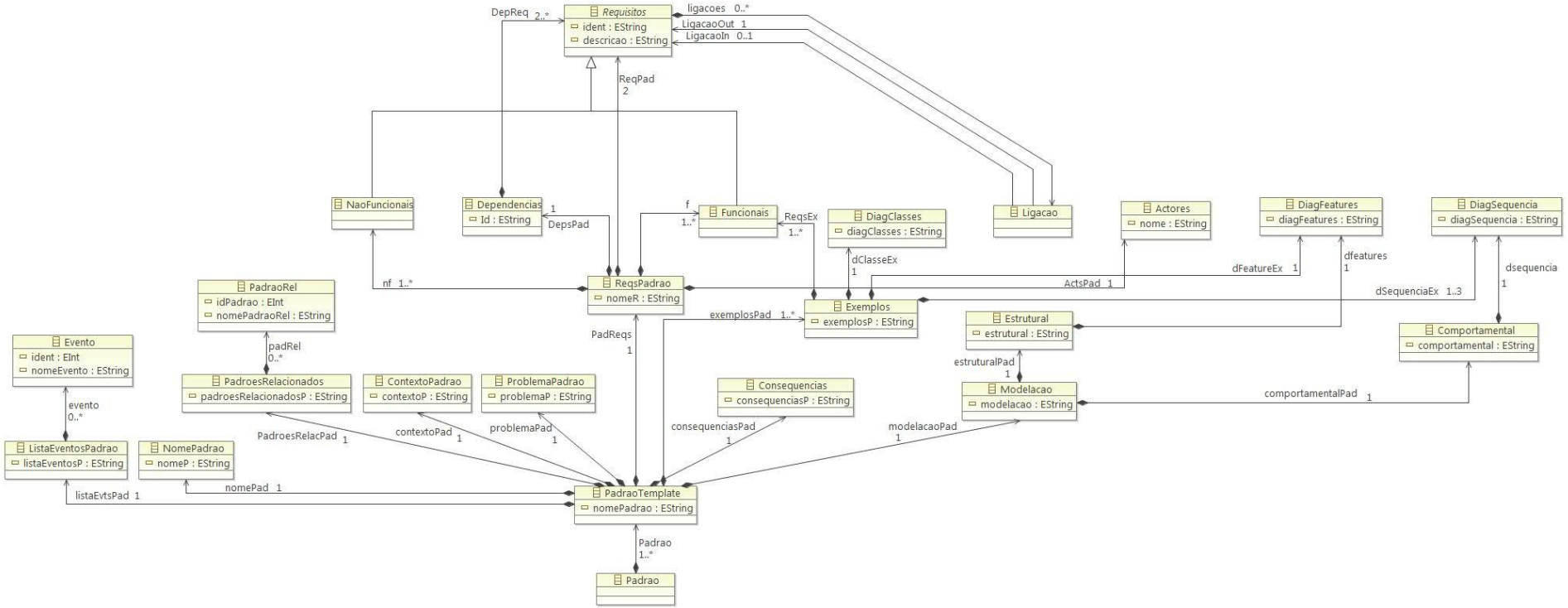

Figura 6.5. Modelo Ecore apenas com as Metaclasses que pertencem ao Editor Principal – Template de Padrão

Abaixo apresenta-se a Tabela 6.2, onde são explicados os conceitos presentes no metamodelo Ecore da Figura 6.6, sendo que a tabela e o metamodelo correspondem ao editor criado para produzir o diagrama de *Features*.

Tabela 6.2. Sub-Editor do Diagrama de *Features*

Elemento	Descrição
DiagFeatures	Representa o diagrama de *Features* do padrão que está a ser descrito. Assim, tem agregados a si todos os campos que compõe o diagrama de *Features*.
Features	Representa as *Features* do padrão.
BinariosFeatures	Representa o conjunto das relações que podem existir entre as *Features* mãe e as filhas, relativamente à sua obrigatoriedade de selecção. Assim, tem agregadas a si as relações *Obrigatorio* e *Opcional*.
Obrigatorio	Representa a propriedade de obrigatoriedade de selecção de uma *Feature*.
Alternativa	Representa uma *Feature* alternativa entre duas *Features* obrigatórias, ou seja, indica que pode ser escolhida mais do que uma *Feature* para o sistema, mas que uma delas tem de ser escolhida.
Or	Representa especialização *or* entre duas *Features* obrigatórias, ou seja, indica que não pode ser escolhida mais do que uma *Feature* para o sistema, mas que uma delas tem de ser escolhida.
Opcional	Representa a propriedade de selecção opcional de uma *Feature*.
OrOpcional	Representa especialização *or* entre duas *Features* opcionais, ou seja, indica que não pode ser escolhida mais do que uma *Feature* para o sistema, mas também não tem de ser escolhida nenhuma obrigatoriamente.
AlternativaOpcional	Representa uma *Feature* alternativa entre duas *Features* opcionais, ou seja, indica que pode ser escolhida mais do que uma *Feature* para o sistema, mas também não tem de ser escolhida nenhuma obrigatoriamente.
DependenciasFeatures	Representa o conjunto das relações de dependência que podem existir entre as *Features*, relativamente à necessidade ou impossibilidade de selecção de uma *Feature* quando outra for seleccionada. Assim, tem agregadas a si as relações *Require* e *Exclude*.
Require	Indica que a *Features* destino deve ser seleccionadas quando a *Feature* origem for escolhida.
Exclude	Indica que a *Features* destino, não podem ser seleccionadas quando a *Feature* origem for escolhida.

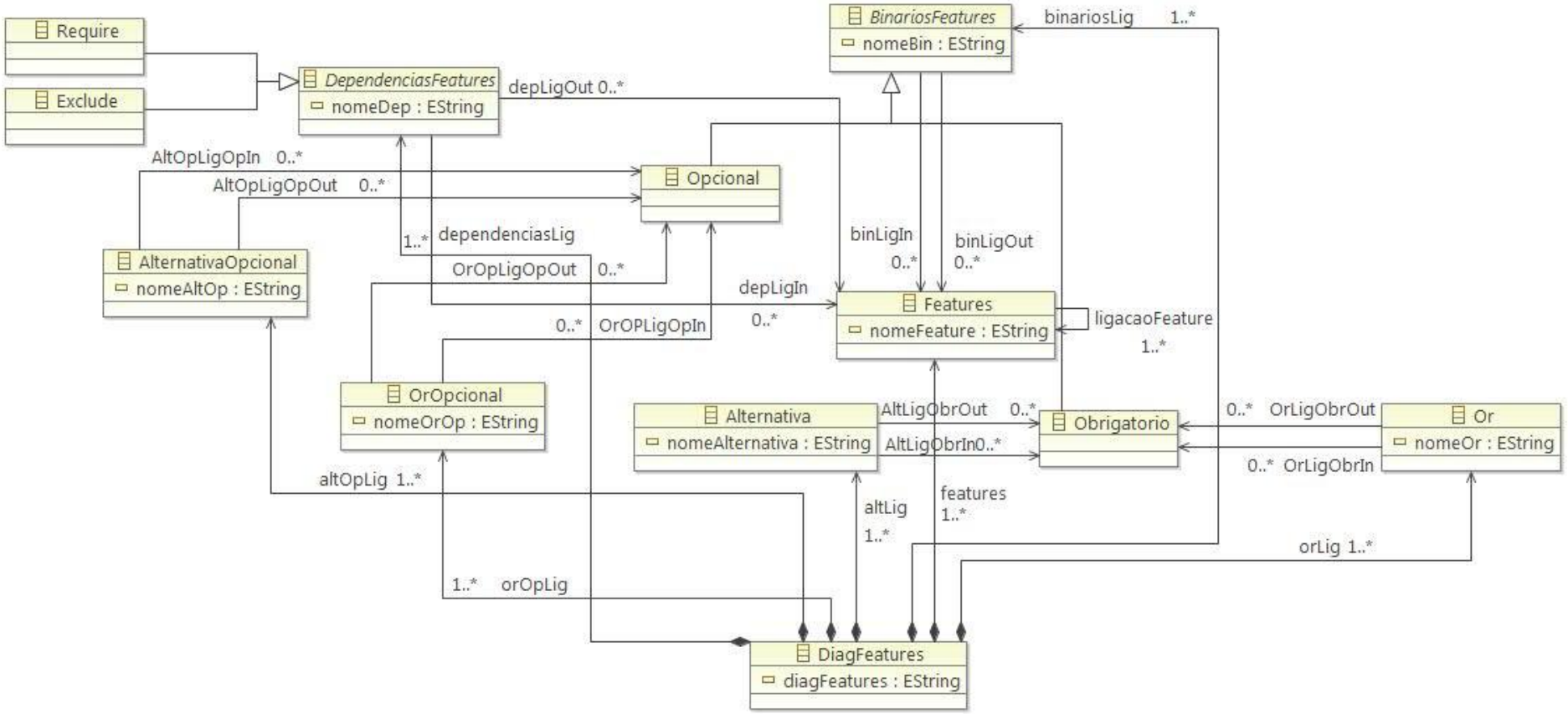

Figura 6.6. Modelo Ecore do Diagrama de *Features*

Abaixo apresenta-se a Tabela 6.3, onde são explicados os conceitos presentes no metamodelo Ecore da Figura 6.7, sendo que a tabela e o metamodelo correspondem ao editor criado para produzir o diagrama de Classes.

Tabela 6.3. Sub-Editor do Diagrama de Classes

Elemento	**Descrição**
DiagClasses	Representa o diagrama de Classes do padrão que está a ser descrito. Assim, tem agregados a si todos os campos que compõe o diagrama de Classes.
Classe	Representa as Classes do padrão.
Atributo	Representa os atributos que pertencem a uma classe do diagrama.
TipoAtributo	Representa o tipo dos atributos pertencentes à classe.
Metodo	Representa os métodos que pertencem a uma classe do diagrama.
ParametrosMetodo	Representa os parâmetros de um método da classe.
TipoParametro	Representa o tipo dos parâmetros de um método da classe
TipoRetornoMet	Representa o tipo de retorno de um método.
Dependencia	Representa o conjunto das relações de dependência que podem existir entre as Classes. Assim, tem agregadas a si as relações de *Agregação* e de *Composição*.
Agregacao	É uma conexão, e representa a relação de Agregação, devendo ser estabelecida entre duas classes.
Composicao	É uma conexão, e representa a relação de Composição, devendo ser estabelecida entre duas classes.
Generalizacao	Representa a relação de Generalização, isto é, representa uma herança existente entre duas classes. Devendo ligar-se, desta forma a classe mãe à classe que dela vai herdar.

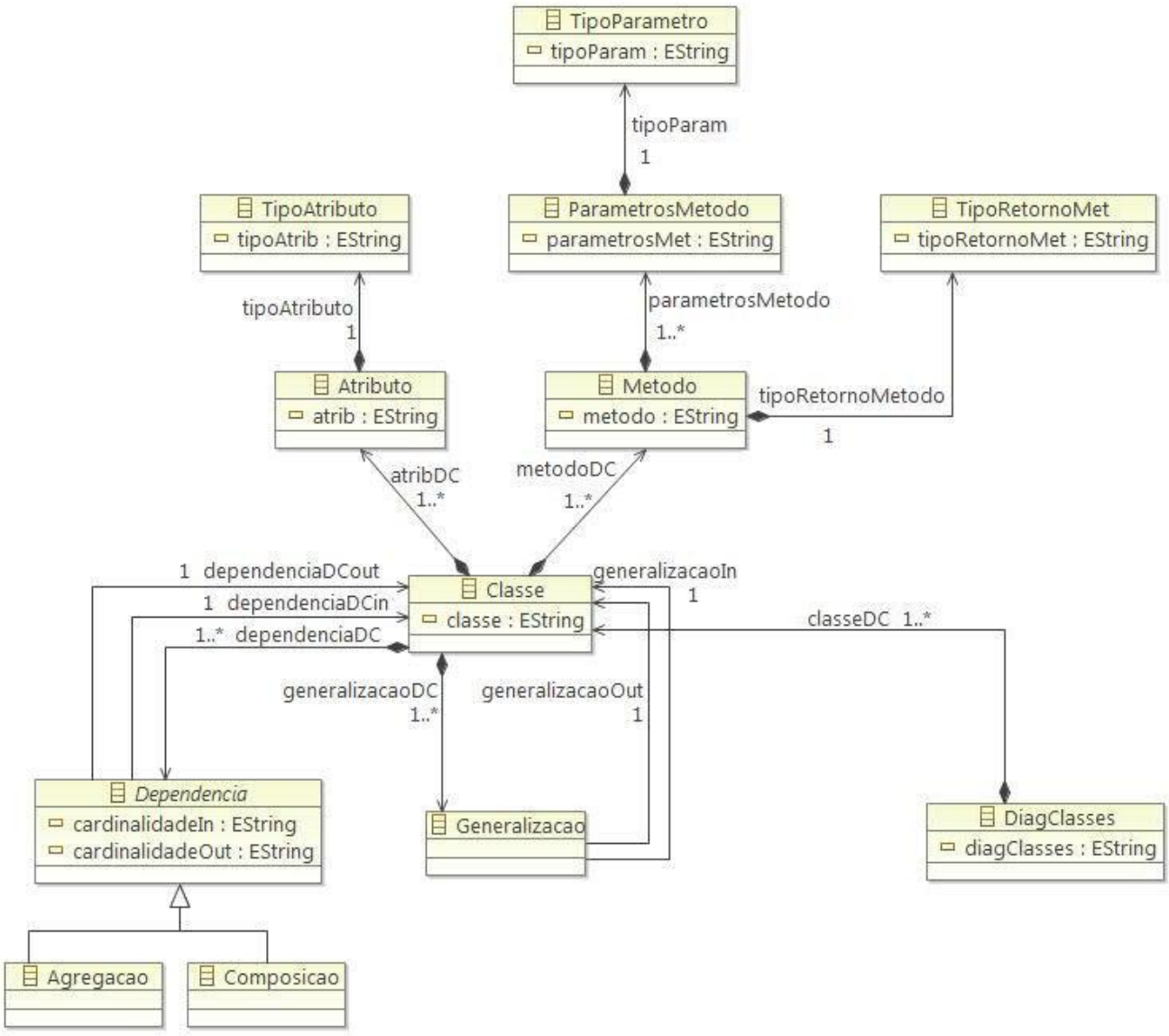

Figura 6.7. Modelo Ecore do Diagrama de Classes

Abaixo apresenta-se a Tabela 6.4, onde são explicados os conceitos presentes no metamodelo Ecore da Figura 6.8, sendo que a tabela e o metamodelo correspondem ao editor criado para produzir o diagrama de Sequência.

Tabela 6.4. Sub-Editor do Diagrama de Sequência

Elemento	**Descrição**
DiagSequencia	Representa o diagrama de Sequência do padrão que está a ser descrito. Assim, tem agregados a si todos os campos que compõe o diagrama de Sequência.
Fragmentos	Representa os compartimentos que podem ser usados num diagrama de Sequência, sendo que o próprio diagrama deve ser construído dentro de um *Fragmento* do tipo *DiagSeq*. Além deste tipo de fragmento, podemos ter fragmentos de *ALT*, *OPT*,

	LOOP, *PAR* e *BREAK*. Finalmente podemos ter ainda o fragmento de *DivFrag* que representa a divisão dos fragmentos que têm condição.
Objectos	Representa os diversos tipos de objecto que podem intervir.
Actor	Representa o objecto do tipo *Actor*.
Interface	Representa o objecto do tipo *Interface*.
Controlo	Representa o objecto do tipo *Controlo*.
Entidade	Representa o objecto do tipo *Entidade*.
LinhaTemporal	Representa o tempo que existe entre os momentos de acção, isto é, o tempo que existe entre os módulos de tempo.
ModulosTempo	Representa o tempo em que há acção no sistema, isto é, o tempo em que há troca de mensagens, chamadas, etc. entre os objectos intervenientes.
Condição	Representa a condição que despoleta um cenário de um fragmento.
Mensagem	Representa uma troca de informação ou um pedido que é efectuado entre dois objectos intervenientes no sistema.
Lig	Esta conexão representa o tempo que antecede alguma acção para um objecto, deve ser estabelecida entre o objecto e o primeiro módulo de tempo.
TipoFrag	Representa um tipo de dados enumerado, este consiste nos diversos tipos de fragmentos que podem existir.
TipoMsg	Representa um tipo de dados enumerado, consiste nos diversos tipos de mensagem que podem existir. Isto é, podemos ter mensagem do tipo *RTN* que representam as mensagens de retorno e as *MSG* que representam os restantes tipos de mensagem.

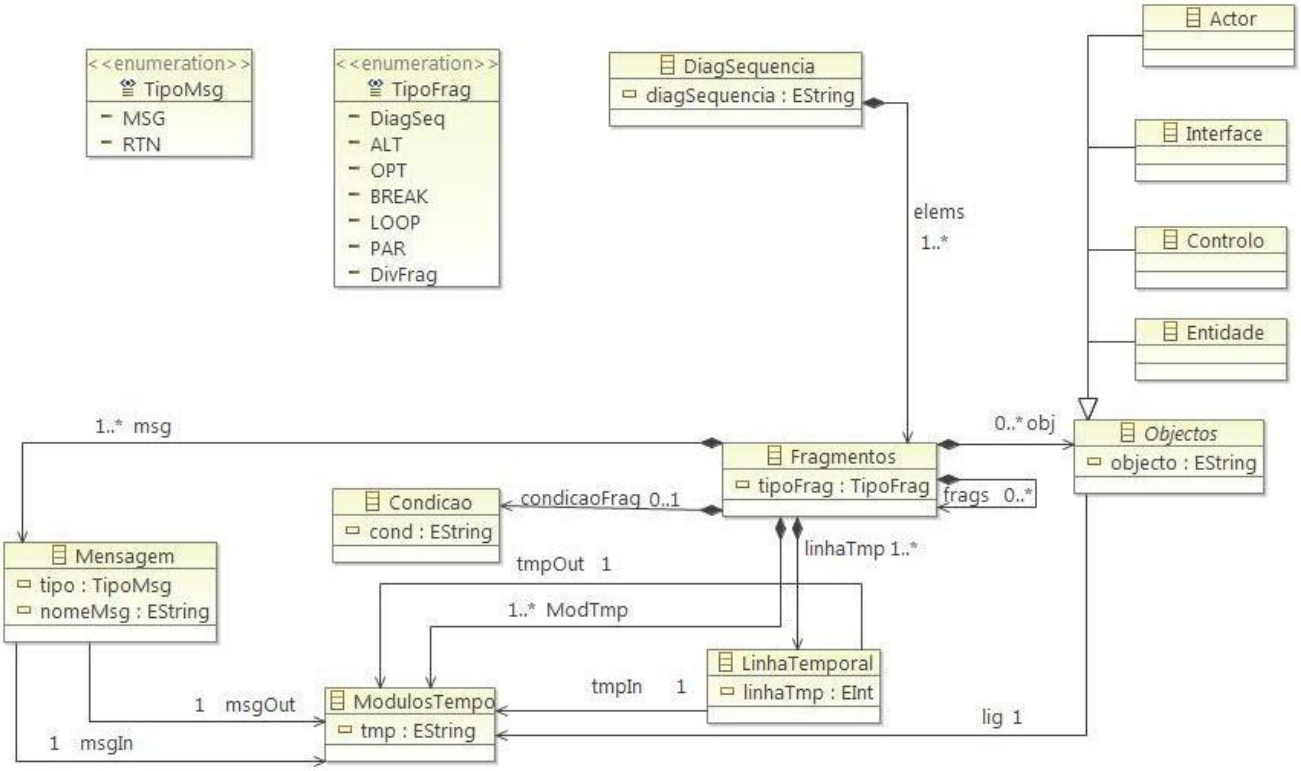

Figura 6.8. Modelo Ecore do Diagrama de Sequência

6.5. Processo de Geração dos Editores

Para gerar os editores começamos por fazer o metamodelo da linguagem usando, como já foi dito, o modelo Ecore. Com este gerou-se o metamodelo Emfatic. Uma vez obtido o ficheiro *.emf*, adicionámos as anotações EuGENia necessárias, com vista a obter o resultado pretendido. Feito isto, para que o metamodelo Ecore ficasse com as anotações do EuGENia adicionadas fez-se a sua geração a partir do Emfatic.

Com o novo modelo Ecore, já se podia começar o processo de geração do GMF. Assim, através do *GMF Dashboard* – Figura 6.9, derivámos o *Domain Gen Model*, isto é o ficheiro *.genmodel*. Através deste ficheiro obtivemos o código do modelo, de edição e do editor. Isto consegue-se abrindo o ficheiro *.genmodel* e clicando com o botão direito do rato sobre a zona da árvore do ficheiro, aqui seleccionam-se as opções *Generate Model Code*, *Generate Edit Code* e *Generate Editor Code*. Depois de gerado o código usa-se o EuGENia para gerar os ficheiros *GMF Tool*, *Graph* e *Map*, para isto clica-se no ficheiro *.ecore* com o botão direito e vai-se a *Eugenia > Generate GMF tool, graph and map models*.

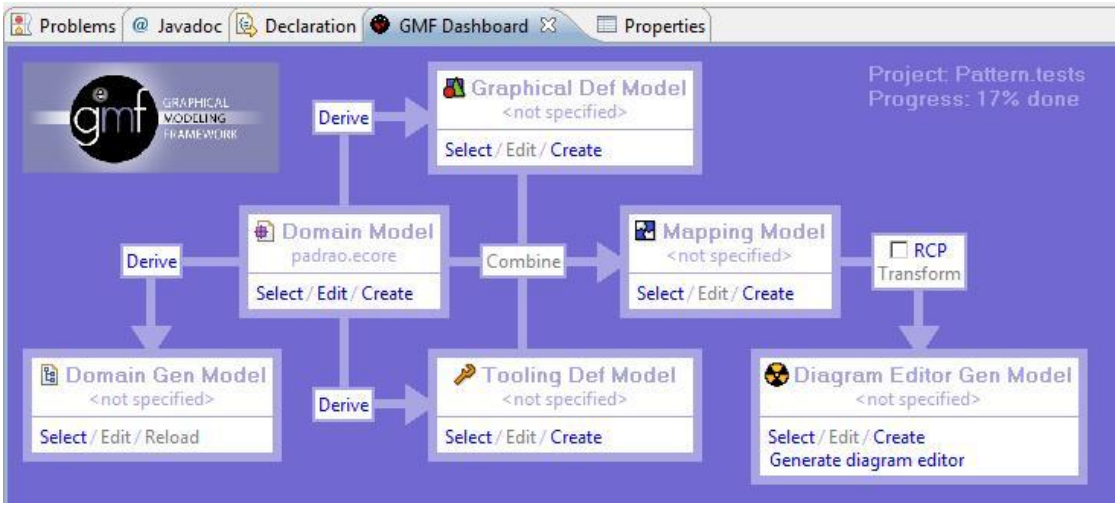

Figura 6.9. GMF Dashboard

Depois, se se pretender fazer alterações, como reordenar a palete ou remover algum elemento, podem editar-se estes ficheiros directamente. No nosso caso, para tornar a palete mais intuitiva no que diz respeito à ordem dos elementos no template, tivemos de fazer uma reordenação dos elementos no *GMF Tool*, ao fazer estas alterações o *GMF Map* fica com as referências para o *GMF Tool* trocadas, e tem de se rectificar estes elementos.

Depois de feitas as alterações pretendidas, estamos prontos para criar o modelo de geração, para tal, clica-se com o botão direito do rato sobre o modelo *GMF Map* e escolhe-se *Create generator Model*. Depois é necessário fazer a sincronização do *Generator Model* com o *GMF genmodel*, para tal clica-se no *Generator Model* com o botão direito do rato e escolhe-se *Eugenia > Synchronize GMF gen model*. Finalmente, resta gerar o código do diagrama, clicando-se com o botão direito do rato sobre o *Generator Model* e escolhendo *Generate diagram code*. Se chegarmos aqui sem erros basta correr como *Eclipse Application* e será apresentada uma nova instância do *Eclipse*. Nesta, fazemos *File>New>Project* e escolhemos *General > Project*. Depois de criado o projecto basta fazer *File>New>Example* e escolher o nome da ferramenta criada – neste caso *Padrao Diagram*.

Esta ferramenta além do editor do *template* do padrão tem 3 sub-editores, um para o diagrama de *Features*, outro para o diagrama de Classes e outro para o diagrama de Sequência. Os metamodelos destes sub-editores são cópia do metamodelo *emf* do editor principal, com a diferença que nestes sub-editores apenas se têm as anotações nas classes que compõem os diagramas e portanto, a metaclasse com a anotação *@gmf.diagram* passa a ser a metaclasse que representa o diagrama.

Assim, no caso do editor principal a metaclasse *Padrao* é a que possui a anotação *@gmf.diagram*, indicando que esta se trata da raiz do metamodelo, no caso do sub-editor do diagrama de *Features* a metaclasse *DiagFeatures*, tem a anotação *@gmf.diagram*, para o sub-editor do diagrama de Classes a metaclasse com a anotação *@gmf.diagram* é a *DiagClasses*, finalmente, no caso do sub-editor do diagrama de Sequência a anotação *@gmf.diagram* está na metaclasse *DiagSequencia*. Estes metamodelos *emf* estão disponíveis nos anexos C, D, E e F respectivamente.

O processo de criação dos sub-editores é idêntico ao de criação de um editor principal. Para a criação do editor principal e dos sub-editores começámos por criar uma pasta para cada um dos editores no projecto GMF. Dentro de cada uma das pastas colocámos o metamodelo *Padrao.emf* com as respectivas anotações e a partir de cada um deles gerámos o metamodelo *Padrao.ecore*. Depois, começando pelos sub-editores, retirámos um de cada vez para fora da pasta e gerámos o *.genmodel* com um nome diferente para cada sub-editor – DF.genmodel; DC.genmodel; DS.genmodel –, seguindo-se a geração do código a partir destes, como descrito no processo geral. Depois gerámos, através do Eugenia o *.gmftool*, *.gmfgraph* e o *.gmfmap*, e a partir do *.gmfmap* gerámos o *.gmfgen*, sendo que este também deverá ficar com um nome diferente para cada sub-editor – DF.gmfgen; DC.gmfgen; DS.gmfgen.

Depois de todos os sub-editores terem sido gerados até ao *.gmfgen*, iniciou-se a geração do editor principal. A geração deste é a habitual até obtermos o *.gmfgraph*, *.gmftool* e o *.gmfmap*. Uma vez gerados estes 3 ficheiros, tivemos de remover do *.gmftool* os elementos que pertencem aos sub-editores, para que estes elementos não apareçam na palete do editor principal. Depois de os removermos do *.gmftool* é necessário rectificar as referências para o *.gmftool* no *.gmfmap*. Feito isto, estamos em condições de fazer a ligação entre os sub-editores e o editor principal. Para tal, foi necessário carregar os *.gmfmap* de cada sub-editor no *.gmfmap* do editor principal, clicando com o botão direito do rato e através da opção *Load Resource*. Feito isto, foi necessário carregar o *.gmfmap* de cada sub-editor no nó do editor principal que conduzirá ao sub-editor. Finalmente, foi necessário editar as propriedades dos elementos que possuem sub-editor no *.gmfgen* do editor principal. Para tal, começamos por ir ao *.gmfgen* do editor principal e em *Gen Editor Generator X.diagram > Gen Diagram XEditPart* seleccionou-se o nó com sub-editor e em *Open diagram behaviour XDiagramEditPolicy*, estão disponíveis as propriedades *Diagram Kind*, que deve ter o *Model ID* que está no nó *Gen Editor Generator X.diagram* do *.gmfgen* do sub-editor. Bem como a

propriedade *Editor ID* que deve ser obtida na propriedade *ID* no nó *Gen Editor View X.diagram.part* do *.gmfgen* do sub-editor.

Feitas as ligações entre os sub-editores e o editor principal foi necessário sincronizar o *.gmfgen* do editor principal para depois se fazer a geração de todos os diagramas.

A visão geral da ferramenta *PatternTool* é apresentada na Figura 6.10. A palete da ferramenta é apresentada no lado esquerdo da figura, onde se podem ver todos os campos necessários para descrever um padrão. Estes campos estão apresentados na palete pela ordem pela qual devem ser adicionados ao *template*. As cores ajudam a compreender a hierarquia existente entre os campos, isto é, que campos devem ser colocados dentro do compartimento de outro.

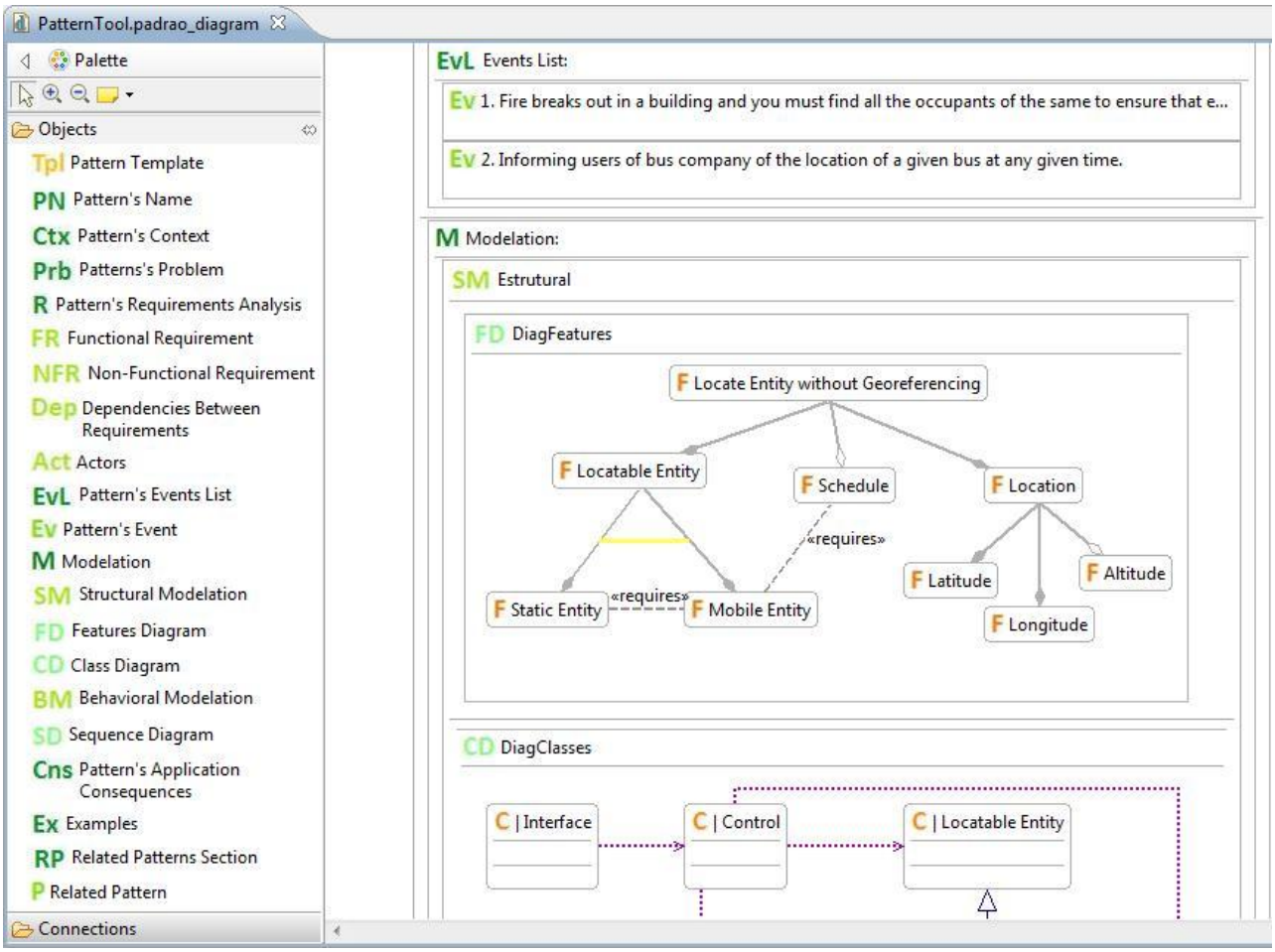

Figura 6.10. Vista Genérica da ferramenta *PatternTool*

O primeiro campo a ser usado deve ser o *Template Pattern* e, portanto o seu *icon* é cor-de-laranja. Os campos que devem ser colocados dentro do compartimento do *Template Pattern* têm um *icon* verde-escuro, como os campos *Pattern's Name*, *Pattern's Requirements Analysis*, entre outros. Quando um campo deve ser colocado dentro do compartimento de um campo com *icon* verde-escuro, o seu *icon* é verde-claro, e finalmente, quando um campo deve

ser colocado dentro do compartimento de um campo com *icon* verde-claro, o seu *icon* deve ser verde ainda mais claro, como é o caso do *icon* do campo *Features Diagram.* Para editar os diagramas de *Features*, Classes e Sequência é necessário adicionar o compartimento do mesmo no *template* e abrir o sub-editor do diagrama correspondente. Para tal, basta clicar duas vezes com o rato sobre o compartimento do diagrama. As paletes dos sub-editores estão organizadas com a mesma ideologia da palete do editor principal. Na Figura 6.11 temos o exemplo de um diagrama de *features* editado no sub-editor para o Diagrama de *Features*.

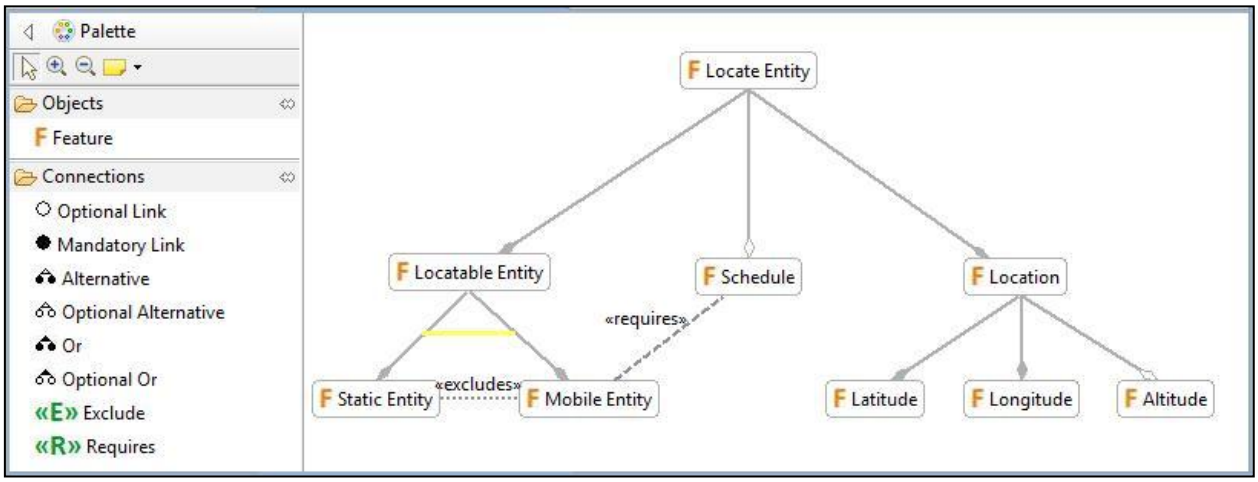

Figura 6.11. Sub-Editor do Diagrama de *Features*

Na Figura 6.12 pode ver-se o sub-editor do Diagrama de Classes e na Figura 6.13 o sub-editor do Diagrama de Sequência.

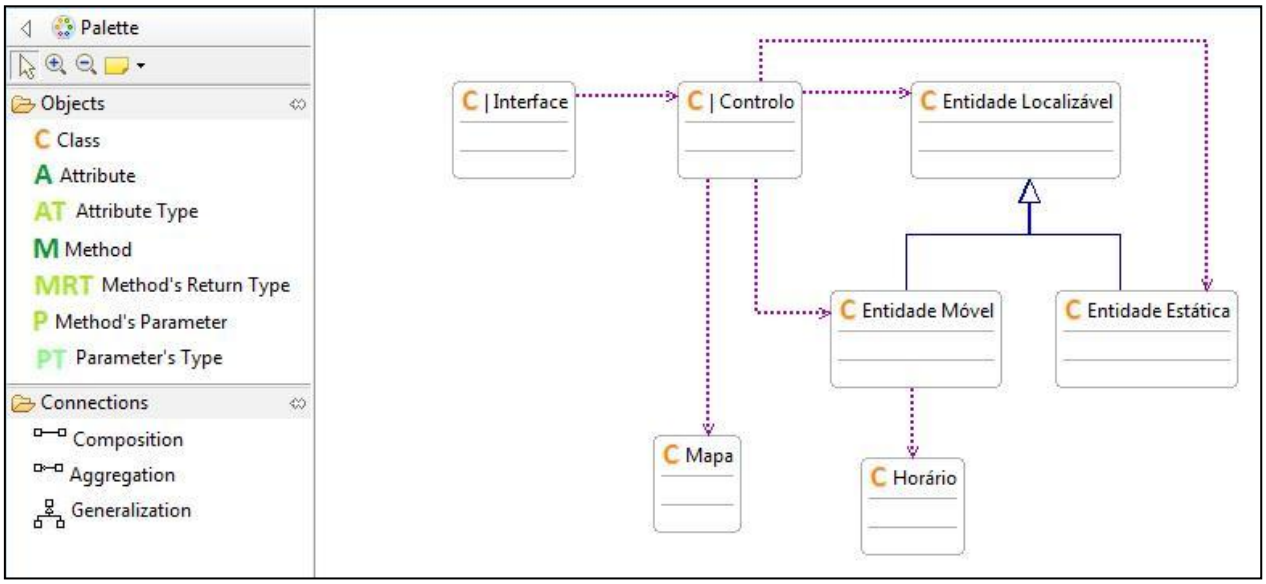

Figura 6.12. Sub-Editor do Diagrama de Classes

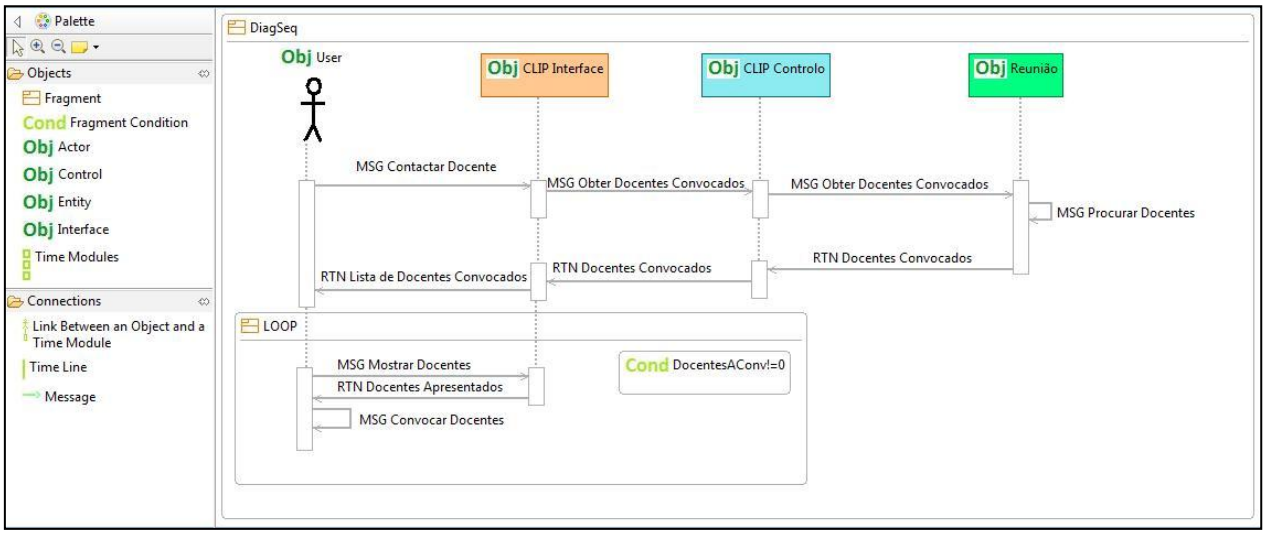

Figura 6.13. Sub-Editor do Diagrama de Sequência

6.6. Sumário

Ao longo deste capítulo apresentámos a ferramenta *PatternTool*, bem como os conceitos e técnicas usadas para a produção da mesma. Tais como, conceitos de Linguagens de Domínio Especifico, a *framework* e os *plugins* usados para o desenvolvimento da ferramenta e ainda todo o processo de desenvolvimento da linguagem.

7. Avaliação

A avaliação efectuada nesta dissertação teve como objectivos avaliar a ferramenta *PatternTool* e a descrição dos padrões propostos na dissertação. Estas avaliações foram conseguidas mediante dois questionários: o primeiro, sobre a ferramenta, em que foram feitas questões relativas à expressividade e sintaxe da linguagem, facilidade de utilização da ferramenta e níveis de satisfação da mesma; e o segundo questionário, sobre a descrição proposta para o padrão, em que foram feitas questões relativamente à simplicidade e clareza da descrição, da pertinência do mesmo e sobre a validade de cada um dos campos descritos no padrão.

Para responder aos questionários foi seleccionado um grupo de quinze pessoas que frequentam ou frequentaram o Mestrado em Engenharia Informática (2º ciclo de Bolonha) na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, com conhecimentos nas áreas de Engenharia de Software, Linguagens de Domínio Específico e Tecnologias de Informação Geográfica.

Assim, pode dizer-se que a avaliação efectuada teve 3 fases importantes:

1. Preparação da Avaliação:
2. Execução da Avaliação;
3. Análise da Avaliação.

Estas fases foram compostas por diversas tarefas. Nomeadamente, na Preparação da Avaliação foi necessário decidir o que era importante ser avaliado, que perfis de utilizadores iríamos contactar, como iríamos conduzir a avaliação – elaborar cenários e questionários. Feito isto foi necessário seleccionar os utilizadores e contactá-los. Já na fase de execução foi necessário combinar com os utilizadores a melhor altura para estes efectuarem a avaliação. Finalmente, com os questionários preenchidos foi feita a análise dos dados obtidos nos questionários, sendo que com estes dados obtivemos gráficos onde se pode ver mais claramente os resultados, que nos permitiu tirar conclusões relativamente à avaliação.

7.1. Questionário para avaliação da ferramenta *PatternTool*

Com vista à avaliação desta ferramenta foi concebido um questionário que foi respondido pelos 15 indivíduos depois destes testarem a ferramenta, usando, como cenário, partes da descrição do padrão *Localizar Espacialmente uma Entidade de uma Aplicação sem Georreferenciação*. O questionário pode ser consultado no Anexo A.

O resultado obtido no questionário foi positivo, uma vez que apesar de alguns utilizadores terem indicado que a ferramenta devia ser melhorada relativamente ao Diagrama de Sequência e que possuía algumas limitações relacionadas com a *framework* usada, no geral, também a consideraram bastante útil, intuitiva, perceptível, fácil de usar e de compreender. Referiram ainda que ajuda muito os elementos estarem na palete pela ordem em que devem ser colocados no *template* e que a ferramenta permite integrar toda a informação para descrever um padrão, o que não existe em outras ferramentas, e é bastante útil por facilitar a consulta e criação das descrições dos padrões.

Abaixo apresenta-se a explicação das diversas questões presentes no inquérito, sendo que algumas mais relevantes serão analisadas com maior profundidade.

7.1.1. Análise do Questionário para avaliação da ferramenta *PatternTool*

L1. Com que facilidade aprendeu os conceitos?

Esta questão pretende avaliar a facilidade com os utilizadores se ambientaram com a linguagem usada na ferramenta. No gráfico da Figura 7.1, pode observar-se que os 15 utilizadores aprenderam os conceitos com facilidade, sendo que destes, 6 consideraram esta tarefa "muito fácil" e 9 consideraram "fácil". Assim, a avaliação dos conceitos foi bastante positiva, indicando este resultado que os conceitos da linguagem são bastante directos e simples de absorver.

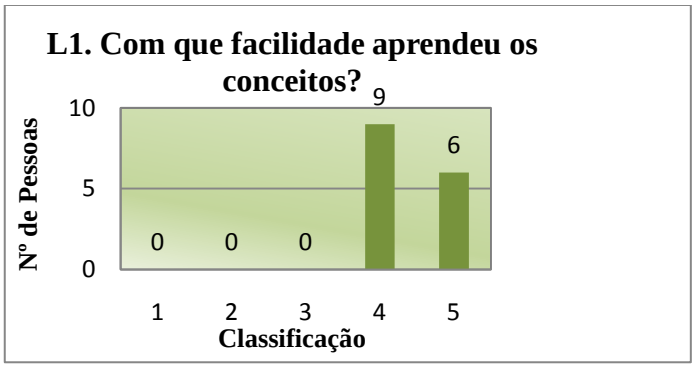

Figura 7.1. Facilidade de aprendizagem dos conceitos

L2. Com que facilidade identificou os símbolos que representavam os conceitos?

Esta questão pretende avaliar a clareza dos símbolos usados na linguagem, para representar os conceitos. No gráfico da Figura 7.2, vê-se que 10 dos 15 utilizadores consideraram que foi "fácil" identificar os conceitos através dos símbolos, 3 consideraram "muito fácil" e 2 deles consideraram a tarefa de dificuldade "moderada". Pode considerar-se a avaliação positiva, pois de uma forma geral os utilizadores consideraram fácil identificar os conceitos. Isto indica que os símbolos escolhidos são bastante intuitivos.

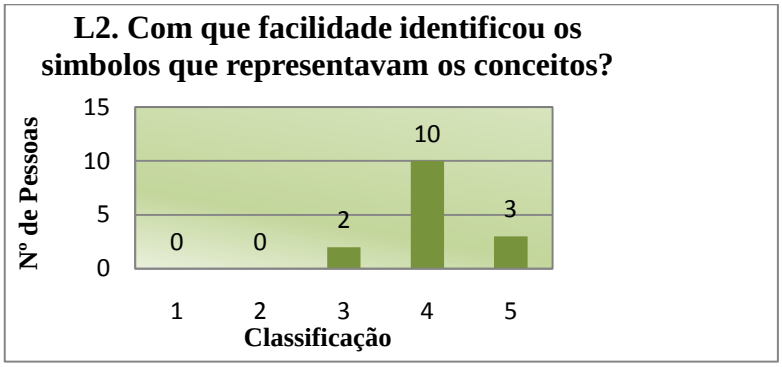

Figura 7.2. Facilidade de identificação dos símbolos que representam os conceitos

L3. Com que facilidade identificou o texto que representava os conceitos?

Esta questão pretende avaliar se o texto que representa os conceitos foi bem escolhido. Pelo que observamos no gráfico da Figura 7.3, 10 dos 15 utilizadores consideraram que foi "muito fácil" identificar os conceitos através do texto, e os restantes 5 consideraram que foi "fácil". Assim, estes resultados indicam que a explicação textual dos conceitos foi bem escolhida, sendo o texto explícito relativamente ao que representa.

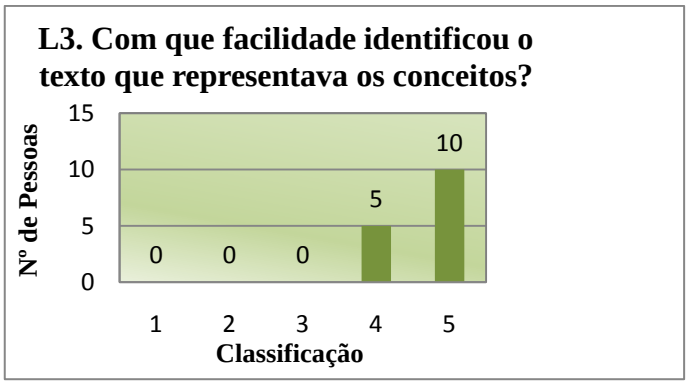

Figura 7.3. Facilidade de identificação do texto que representa os conceitos

L4. Com que frequência cometeu erros devido à semelhança dos símbolos?

Esta questão foi feita com a intenção de reforçar a avaliação da escolha dos *icons*, desta vez para verificar se estes têm alguma parecença que possa induzir os utilizadores em erro. Como podemos ver no gráfico da Figura 7.4, os resultados foram bastante positivos e muito coerentes com os resultados anteriores, uma vez que apenas 1 utilizador indicou ter cometido erros deste tipo com uma frequência "moderada" e os restantes consideraram ter cometido erros com "pouca" – 4 utilizadores – ou "nenhuma" – 10 utilizadores – frequência.

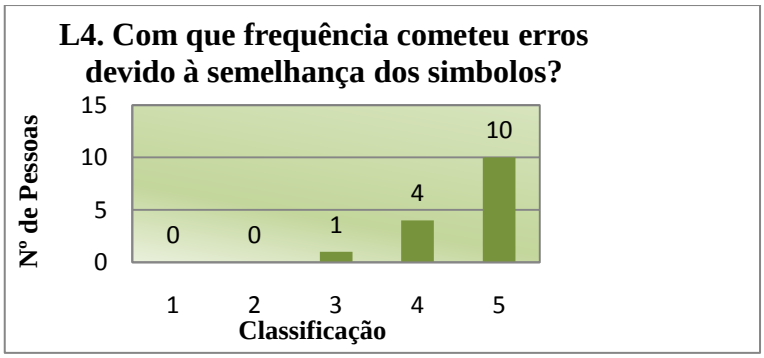

Figura 7.4. Frequência de erros devido à semelhança de símbolos

L5. Com que frequência cometeu erros devido à ambiguidade do vocabulário?

Esta questão foi feita para identificar alguma falha de vocabulário que poderia ter sido identificada pelos utilizadores. Como podemos ver pelo gráfico da Figura 7.5, apenas 2 utilizadores cometeram erros com frequência "moderada" por ambiguidade de vocabulário, sendo que dos restantes 13, 3 consideraram ter cometido erros com "pouca" frequência e 10 consideraram mesmo não ter cometido "nenhum" erro por ambiguidade de vocabulário. Assim, concluímos que em termos de vocabulário o editor é bastante claro.

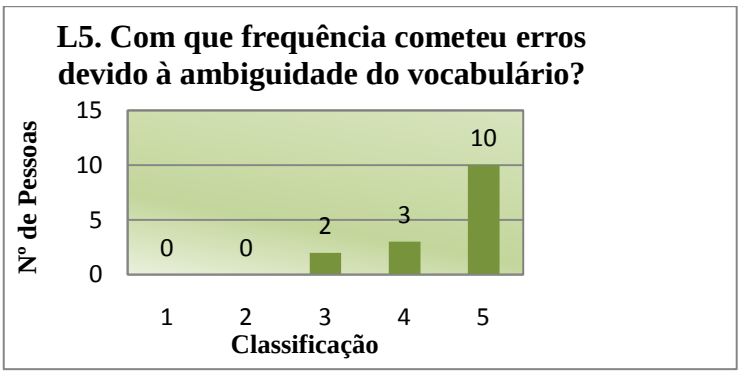

Figura 7.5. Frequência de erros devido a ambiguidade de vocabulário

L6. Com que frequência se sentiu incapaz de expressar o que desejava?

Esta questão pretende verificar se há algum conceito que esteja a falhar na linguagem. Concluímos que os conceitos da linguagem de um modo geral são claros, uma vez que apenas 1 utilizador se sentiu incapaz de expressar o que desejava de forma "moderada", sendo que 8 deles se sentiu "pouco incapaz" de se expressar e os restantes 6 utilizadores "nunca" se sentiram incapazes de se expressar, como podemos ver no gráfico da Figura 7.6.

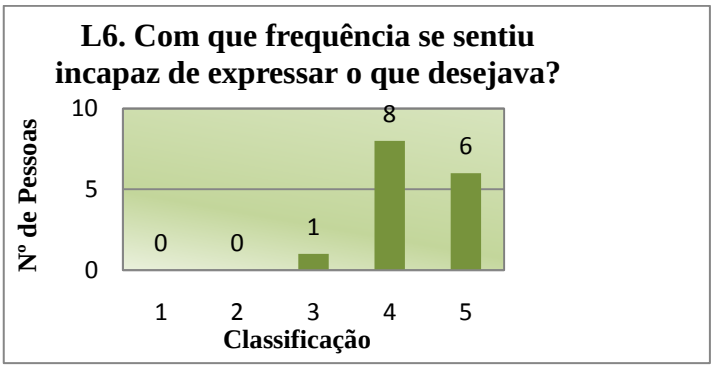

Figura 7.6. Frequência de incapacidade para expressar o pretendido

Elaboração do Cenário

Este grupo de questões – de C1 a C7 – foi feito com o propósito de verificar como o utilizador se comporta e se sente, perante um caso concreto, em que tenha de usar a ferramenta, para completar um processo.

C1. Com que frequência sentiu necessidade de consultar a documentação durante a execução do cenário?

C2. Com que frequência teve necessidade de questionar o supervisor durante a execução do cenário?

C3. Como classifica o seu grau de confiança durante a execução do cenário dado?

C4. Qual a sua impressão geral da ferramenta?

C5. Como classifica a dificuldade do cenário?

C6. O que considerou mais difícil de fazer?

C7. Como se sente relativamente à correcção do cenário elaborado?

A avaliação feita através das questões C1 a C7 foi positiva, pois na maioria dos casos os utilizadores tiveram um bom desempenho, sendo que apenas alguns se sentiram um pouco confusos em algum momento durante a execução do cenário.

Usabilidade

Tempo gasto a elaborar o template do Padrão fornecido. ___minutos.

Esta questão tinha o objectivo de testar o tempo médio necessário para a 1ª utilização da ferramenta, quando usada para completar uma descrição. Verificou-se que em média os utilizadores demoraram 85 minutos a completar o exercício.

F1. Qual a sua impressão geral da ferramenta?

Esta questão pretende avaliar se os utilizadores, de forma geral, gostaram de usar a ferramenta. Pelo gráfico da Figura 7.7 vemos que 12 dos 15 utilizadores consideraram que a ferramenta é "boa", 2 consideraram "muito boa" e 1 deles considerou "mediana". Isto demonstra que de uma forma geral os utilizadores que usaram a ferramenta gostaram de a usar.

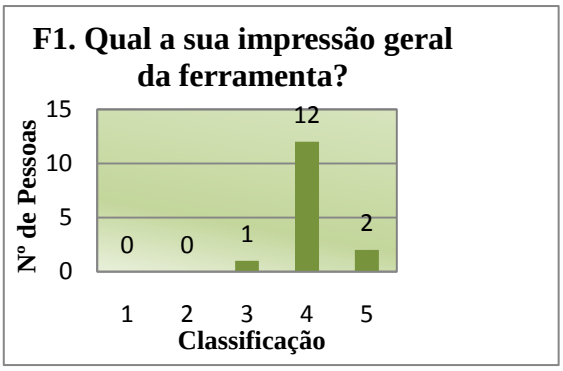

Figura 7.7. Impressão geral da ferramenta

F2. Como se sentiu ao fazer alterações?

Esta questão foi feita para avaliar a facilidade de edição do padrão, com vista a saber se quando, no futuro, for necessário alterar um padrão já criado, se seria fácil fazê-lo. O resultado desta avaliação foi positivo, uma vez que, como podemos ver no gráfico da Figura 7.8, 11 dos 15 utilizadores dizem ter-se sentido "confiantes" ao fazer alterações, 3 dizem ter-se sentido "muito confiantes" e apenas 1 se sentiu confiante de forma "moderada".

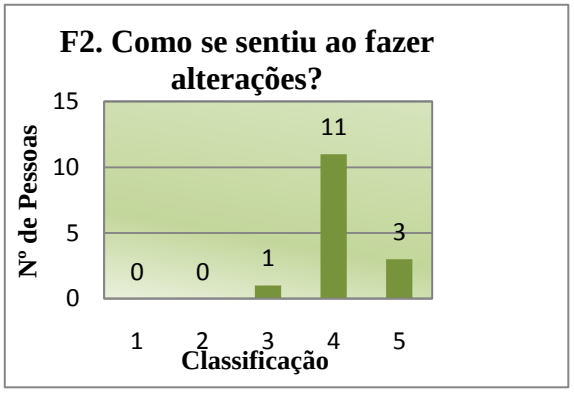

Figura 7.8. Confiança ao fazer alterações

F3. Considera que foi fácil passar o padrão do papel para a ferramenta?

Esta questão pretende avaliar se os utilizadores se sentiram perdidos a fazer a passagem do padrão do papel para a ferramenta. Pelo gráfico da Figura 7.9 vemos que 11 dos 15 utilizadores consideram "fácil" e os outros 4 consideram "muito fácil". Isto demonstra que a ferramenta segue o *template* seleccionado, e usado na descrição do padrão em papel, sendo praticamente igual preencher o *template* no papel ou usando a ferramenta, havendo a vantagem de que, usando a ferramenta, a descrição fica em formato electrónico.

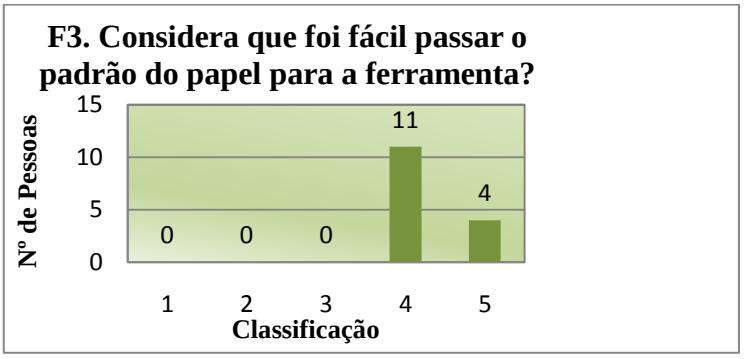

Figura 7.9. Facilidade de passagem do papel para a ferramenta

F4. O resultado final do padrão, na ferramenta, era o que estava à espera?

Esta questão avalia se os utilizadores consideraram que a ferramenta difere muito do *template* usado para produzir a descrição em papel. Pelo gráfico da Figura 7.10 vemos que dos 15 utilizadores apenas 1 deles "Não estava à espera" do resultado que obteve. O resultado é bastante positivo, pois demonstra que a ferramenta segue o *template* do padrão, tornando-se bastante intuitiva.

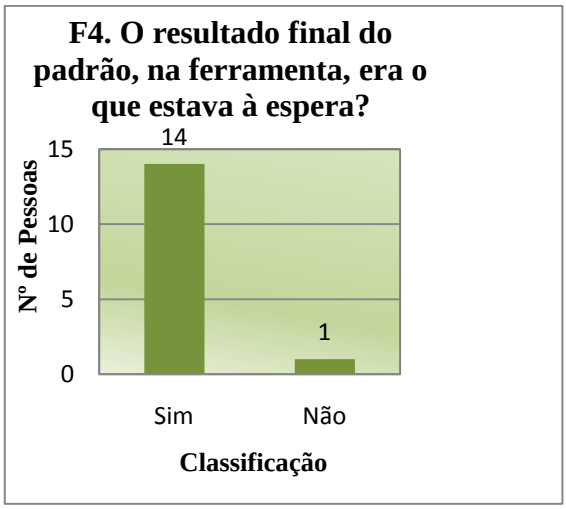

Figura 7.10. Resultado Final na Ferramenta é o Esperado

F5. Que alterações propunha para a ferramenta?

Esta questão tinha a intenção de obter algumas sugestões para melhorar a ferramenta. Algumas destas sugestões são citadas de seguida:

- "Penso que a parte mais complicada será nos diagramas de sequência, com a introdução de fragmentos tipo ALT, OPT, etc. Essa parte talvez não seja tão intuitiva uma vez que para criar um fragmento temos de introduzir um de divisão e outro de condição";
- "Impedir a repetição de *Idents* nos requisitos. Embora os Idents sejam *strings*, fazer alguma validação do seu formato"*;*
- "Na dependência dos Requisitos, poderia arranjar-se uma forma de arrastar os requisitos que já foram especificados em **FR e NFR**, e ligá-los, evitando a sua reinserção";
- "Introdução de código no editor para produzir um padrão mínimo automaticamente";

- "No diagrama de sequência, é permitida a ligação em grafo circular das *timelines*, isto poderia ser resolvido com OCL";

- "No diagrama de *features* permite dependências circulares de *features*, os sentidos não são verificados e permite múltiplas ligações entre as mesmas duas *features*, mais uma vez poderia ser resolvido com OCL".

F6. Compreendeu o processo de criação do Padrão?

Esta questão pretende avaliar se perante a ferramenta e com o *template* em papel o utilizador conseguiu produzir com facilidade o resultado pretendido. Pelo gráfico da Figura 7.11, vemos que 100% das respostas foram "Sim", o que mostra, mais uma vez, que não há grande complexidade no processo de utilização da ferramenta.

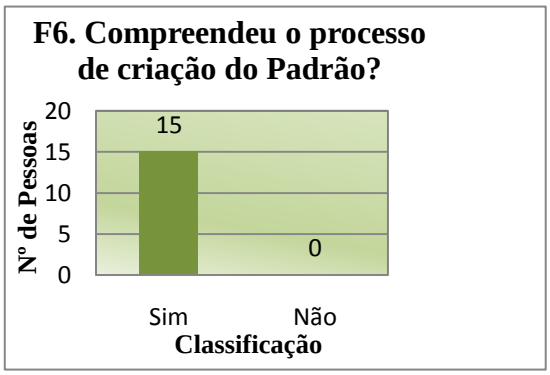

Figura 7.11. Compreensão do Processo de Criação do Padrão

F7. Com que facilidade executou o processo em F6?

Esta questão pretende avaliar a facilidade do processo de criação do padrão. Pelo gráfico da Figura 7.12, vê-se que 10 dos 15 utilizadores, além de compreenderem o processo, também tiveram facilidade em executá-lo, e que outros 3 tiveram "grande facilidade" em executá-lo, no entanto, 2 utilizadores, apesar de compreenderem o processo, consideraram "mediana" a dificuldade de execução do processo. De um modo geral, pode dizer-se que o processo de criação é simples. No entanto, para os utilizadores menos experientes na utilização de LDEs, pode ser mais difícil executar o processo.

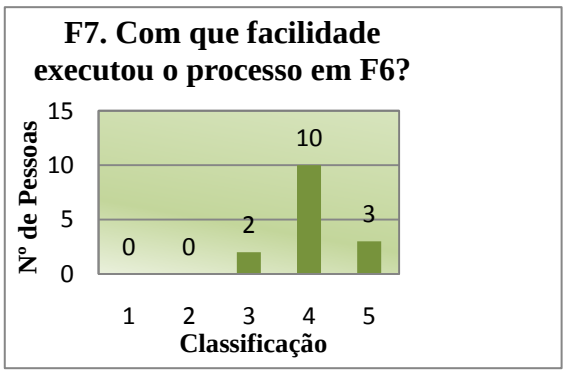

Figura 7.12. Facilidade de Execução do Processo de Criação do Padrão

F8. Teve dificuldades a utilizar a ferramenta?

Esta questão foi feita com o intuito de identificar possíveis actividades mais complexas do processo, com vista a poder tentar simplificá-las. Como vemos pelo resultado apresentado no gráfico da Figura 7.13, na generalidade, os utilizadores não tiveram dificuldades em usar a ferramenta. No entanto, 1 utilizador demonstrou ter sentido alguma dificuldade. De qualquer das formas, pode considerar-se que o resultado é bastante positivo.

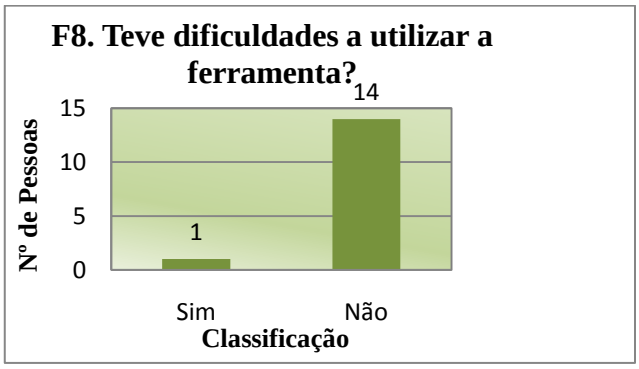

Figura 7.13. Dificuldades na utilização da ferramenta

F9. Comparando a descrição do padrão que lhe foi fornecida em papel e o gerado na ferramenta, como classifica o resultado esperado?

Esta questão pretende verificar a satisfação do utilizador perante o resultado obtido. Que como podemos ver pelos resultados apresentados no gráfico da Figura 7.14, foi bastante satisfatória, uma vez que dos 15 utilizadores apenas 1 considerou o resultado obtido relativamente ao que esperava "mediano", 10 dos restantes utilizadores consideraram o resultado "semelhante", e os restantes 4 consideraram-no "muito semelhante".

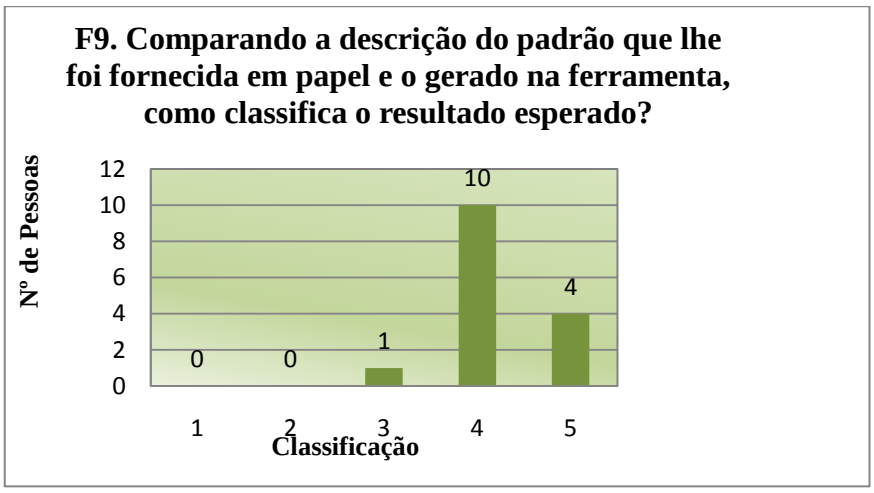

Figura 7.14. Classificação do resultado obtido relativamente ao esperado

F10. Ao mudar de um sub-editor (Diagrama de Classes, de Features ou de Sequência) para o editor principal (Template), denotou alguma perda de informação?

Esta pergunta foi feita para verificar se havia alguma falha não detectada nos sub-editores. Pelo gráfico da Figura 7.15, vemos que 11 dos utilizadores consideraram que não há perda de informação e que os restantes 4 consideraram que há. No entanto, por observação dos testes, constatou-se que esta perda foi identificada por utilizadores que desconheciam as limitações da *framework* EMF/GMF no que respeita à preservação da posição de um elemento quando se passa de um sub-editor para o editor principal. Assim, os utilizadores criaram os diagramas de *Features,* Classes e de Sequência e colocaram os elementos com a disposição adequada, mas quando salvaram e voltaram ao editor principal os elementos estavam todos na mesma posição sobrepostos. Isto não se trata de uma limitação da ferramenta em si, mas sim da *framework* utilizada. Na verdade, não é perdida nenhuma informação, o que se perde é a posição dos diferentes elementos. Desta forma, pode concluir-se que não foi detectada nenhuma perda de informação por parte dos utilizadores.

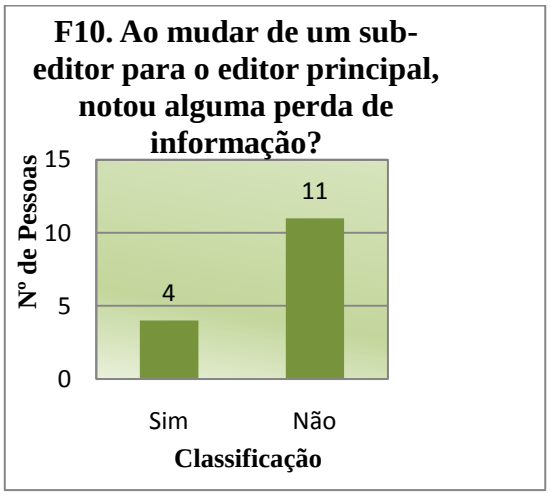

Figura 7.15. Perda de informação na passagem entre editores

F11. Como classifica a ferramenta comparando com outras ferramentas com que já trabalhou?

Esta questão pretende avaliar se a ferramenta criada tem vantagens relativamente a outras com propósitos semelhantes. Pelo gráfico da Figura 7.16 vemos que dos 15 utilizadores, 3 deles "nunca usaram nenhuma ferramenta semelhante", 7 deles consideraram a ferramenta "mediana" relativamente a outras, 4 consideraram-na "boa" e 1 considerou-a "muito boa". Este resultado mostra que apesar dos utilizadores considerarem a ferramenta boa, de um modo

geral, quando comparada com outras ferramentas mais focadas em construção de diagramas, notam que a presente tem algumas limitações, ainda que algumas dessas limitações tenham a ver directamente com a *framework* usada.

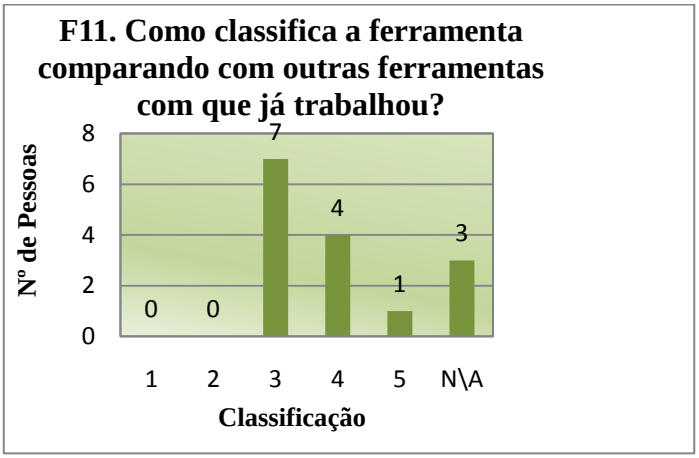

Figura 7.16. Comparação da Ferramenta com outras semelhantes

F12. Considera a ferramenta útil?

Esta questão pretende avaliar se os utilizadores consideraram que a ferramenta é benéfica e, se faz sentido a sua criação. Como podemos ver na Figura 7.17, todos os utilizadores consideraram que "sim". Isto demonstra que apesar de, quando comparada com outras, ter aspectos menos bons, é útil. Na maioria dos casos a justificação é que nenhuma ferramenta junta a criação do *template* de padrão com a criação de diagramas de *Features*, de Classes e de Sequência.

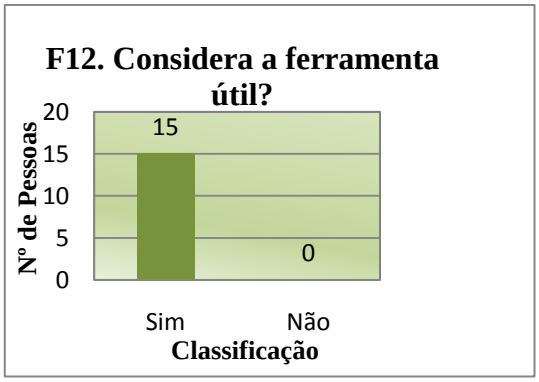

Figura 7.17. Utilidade da Ferramenta

As questões que se seguem – de F13 a F18 – são de carácter mais textual e foram colocadas com o objectivo de identificar os pontos mais positivos e mais negativos da ferramenta, com vista a perceber o que se torna mais e menos confortável para os utilizadores. Desta forma

poderemos, se necessário, proceder a algumas alterações de forma a tornar a ferramenta mais fácil de usar e mais aliciante. Apresentam-se abaixo as citações das sugestões/opiniões de alguns dos utilizadores.

F13. O que acha que a descrição obtida na ferramenta traz de melhor em relação à que lhe foi fornecida em papel?

- "Interactividade, facilidade de correcção e evolução do padrão, o *output* do editor poder ser integrado com outras ferramentas, para verificação e transformação. O editor de padrões ao evoluir pode fornecer uma assistência maior ao processo de criação";
- "Possibilidade de síntese, agrupamento lógico visual e hierárquico (muito útil)";
- "A descrição do padrão através da ferramenta torna mais simples a leitura do próprio padrão, uma vez que se consegue ter uma visão global de todos os elementos que constituem a sua descrição".

F14. E o que acha que traz pior?

- "Sendo o padrão condensado todo num *template*, fica muita informação compactada no mesmo sítio, estando tudo descolapsado o diagrama fica grande. Impressões em papel podem estar comprometidas, a menos que sejam feitas de forma segmentada".

F15. Qual a maior dificuldade que encontrou no uso da ferramenta?

- "Penso que as dificuldades encontradas foram as normais e esperadas, dado que foi a primeira vez que utilizei a ferramenta. Encontrei algumas dificuldades na criação de diagramas de sequência, mas que foram rapidamente eliminadas, à medida que fui tomando conhecimento da ferramenta";
- "Assistência na produção de padrões como não *expert* do domínio";
- "Limitações de expansão do tamanho dos compartimentos impostas pelo GMF".

F16. Quais são, na sua opinião, os pontos fracos desta ferramenta?

- "A falta de algumas verificações, e alguma modelação ambígua".

F17. E quais pontos fortes da ferramenta?

- "De uma forma geral permite uma modelação flexível e extensiva de padrões";
- "A ferramenta é robusta, e tal como dito anteriormente, é de fácil utilização, e condensa muito bem a informação sobre o padrão";
- "Hierarquia visual. Disponibiliza como que um fio condutor para criação do padrão";
- "A possibilidade de criar os diagramas relacionados com o padrão, na mesma aplicação, poder ter acesso a toda informação sobre o padrão, num mesmo ecrã".

F18. O que sugere para que a ferramenta possa ser melhorada?

- "A introdução de código OCL e código costumizado para verificação de algumas secções do padrão";
- "Introdução de código para:
  - Inicialização mínima do padrão e subelementos;
  - Geração de identificadores únicos onde necessário";
- "Introdução de valores *default* onde aplicável, como por exemplo cardinalidades de ligações, pesos, etc.".

7.2. Questionário para avaliação da descrição do Padrão

Com vista à validação da descrição feita para o padrão *Localizar Espacialmente uma Entidade de uma Aplicação sem Georreferenciação,* foi efectuado um questionário que foi respondido pelos 15 indivíduos, depois de analisarem a descrição que lhes foi fornecida do padrão *Localizar Espacialmente uma Entidade de uma Aplicação sem Georreferenciação*. O questionário pode ser consultado no Anexo B.

Apesar de apresentarmos a descrição para dois padrões, nesta avaliação apenas se considerou o padrão relativo à localização porque, durante a fase de avaliação, o padrão de apresentação ainda não estava terminado, além de que a estrutura do padrão e a forma como este foi pensado e descrito segue a ideologia do primeiro.

O resultado obtido no questionário efectuado foi positivo, uma vez que os utilizadores consideraram que o padrão descrito é bastante relevante e útil, que a descrição é clara e simples e aplicável a diversas áreas e fácil de reutilizar. Os utilizadores consideraram ainda que a introdução dos cenários é bastante útil para ajudar na modularização. Como pontos mais negativos apenas apontaram o facto do nome do padrão ser bastante comprido e que as dependências deviam estar representadas num diagrama, de forma a ser mais imediata a sua leitura.

Abaixo apresenta-se a explicação das diversas questões presentes no inquérito, sendo que algumas serão analisadas de forma mais aprofundada.

7.2.1. Análise do Questionário para avaliação da descrição do Padrão

1. Como classifica a relevância do padrão proposto?

Esta questão pretende avaliar se o utilizador considera que o padrão é, de facto, importante e poderá vir a trazer benefícios para outras aplicações. De acordo com os resultados apresentados no gráfico da Figura 7.18, vemos que apenas 3 dos 15 utilizadores consideraram a relevância "moderada", e que 11 deles consideraram "alta", sendo que houve 1 que considerou mesmo "muito alta". Assim, concluímos que de um modo geral o padrão foi considerado importante e de grande utilidade.

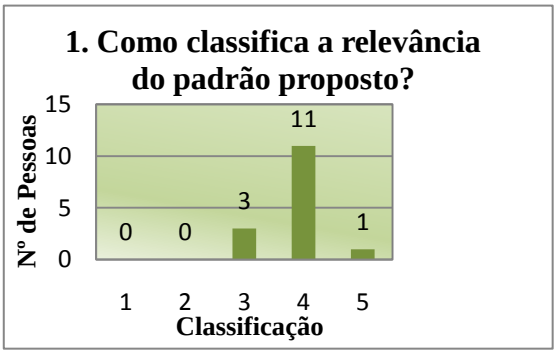

Figura 7.18. Relevância do Padrão Proposto

2. Acha que este pode ser reutilizado em várias aplicações de diversas áreas?

Esta questão foi feita para avaliar se os utilizadores consideraram que o padrão está suficiente bem descrito e genérico para ser aplicado a diversas áreas e em diversas aplicações. Como é visível no gráfico da Figura 7.19, os utilizadores consideraram todos que o padrão poderá ser aplicado em várias aplicações e de diversas áreas, o que é bastante positivo, uma vez que se pretendia que o padrão pudesse ser aplicado a aplicações de todas as áreas que pudessem beneficiar de georreferenciação e que não a tivessem.

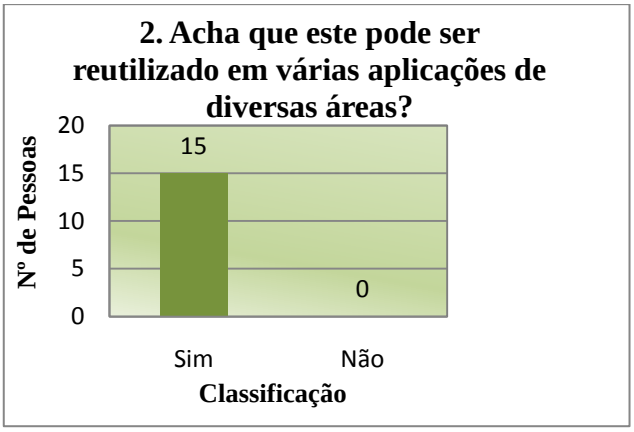

Figura 7.19. Reutilização do Padrão

3. Acha que as aplicações SIG podem beneficiar deste padrão?

Esta pergunta foi feita para corroborar a importância do padrão, desta vez, verificando se os utilizadores consideram que a sua utilização pode trazer benefícios para as aplicações de Sistemas de Informação Geográfica, e mais uma vez, todos os utilizadores demonstraram acreditar que o padrão pode trazer benefícios para este tipo de aplicação, como se pode ver no gráfico da Figura 7.20.

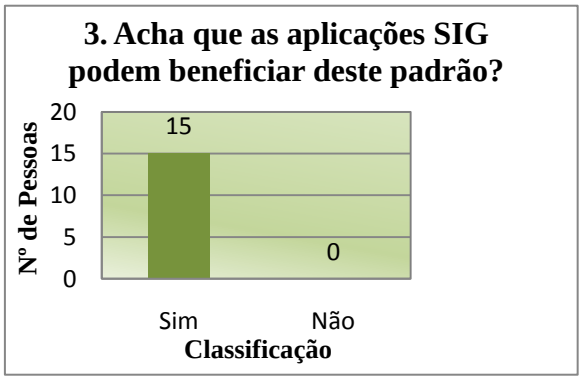

Figura 7.20. Beneficio deste Padrão

4. Como classifica a clareza e simplicidade da definição do padrão?

Esta questão pretende verificar se a definição do padrão está escrita de forma simples e clara, com vista a poder melhorar a descrição de acordo com a opinião dos utilizadores. No entanto, as respostas dos utilizadores foram bastante positivas, na medida em que 13 dos 15 utilizadores consideram que relativamente à clareza e simplicidade o padrão está "bom" e os restantes 2 consideraram mesmo que está "muito bom", como se pode ver no gráfico da Figura 7.21.

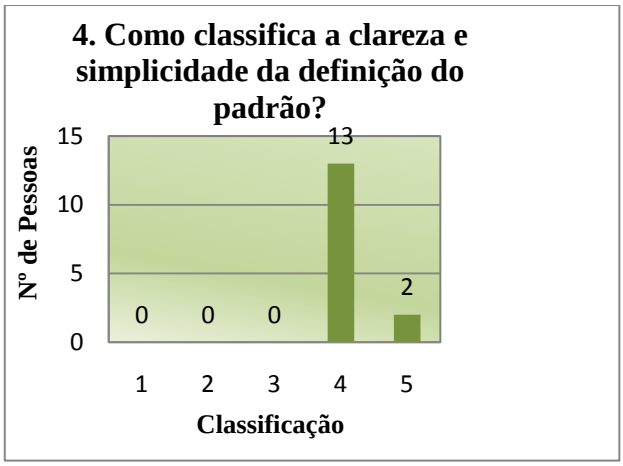

Figura 7.21.Clareza e Simplicidade da Definição do Padrão

As questões que se seguem, da questão 5 à 15, têm o objectivo de obter, da parte dos utilizadores, sugestões relativamente ao nome, definição dos requisitos, eventos, e todos os outros campos do padrão, por forma a poder melhorar a descrição produzida e a perceber que tipo de informação se torna mais ou menos clara e perceptível para os utilizadores. O resultado obtido com estas questões foi bastante positivo uma vez que alguns utilizadores fizeram sugestões bastante interessantes ao nível de reduzir o nome do padrão, e de alguns eventos que poderiam despoletar o padrão. Relativamente à correcção do que estava já descrito, nenhum utilizador apontou nada como menos correcto. E todos os requisitos, eventos, etc. já descritos foram consideradas importantes, por parte dos mesmos.

5. Como classifica a escolha do nome do Padrão?

6. Tem alguma sugestão melhor para o nome do Padrão?

- "Localizar Espacialmente Entidades sem Georreferenciação"

7. Acha que a introdução da lista de eventos é importante para quem vier a reutilizar este padrão?

8. Como classifica a pertinência dos eventos apontados na lista do padrão?

9. Acrescentaria ou removeria algum evento?

Um utilizador, nesta questão sugeriu que se acrescentasse um evento para localizar um técnico de assistência, por exemplo, da Zon, permitindo que a empresa fizesse uma melhor gestão dos recursos, isto é se é necessário um técnico em determinada zona, poderia localizar-se os técnicos que estão em trabalho de forma a ver qual o que estava mais próximo da zona onde era necessário o técnico.

10. Como classifica a pertinência dos requisitos funcionais descritos no padrão?

11. Acrescentaria ou removeria algum requisito funcional?

12. Como classifica a pertinência dos requisitos não-funcionais descritos no padrão?

13. Acrescentaria ou removeria algum requisito não-funcional?

14. Como classifica a pertinência das dependências entre os requisitos do padrão?

15. Alteraria alguma dependência?

16. Considera que a introdução do diagrama de features é útil?

A questão 16 tem como objectivo perceber se os utilizadores se sentiram confortáveis com a introdução do diagrama de *Features* na descrição do padrão e se consideraram a informação dada por este útil. Como vemos pelo gráfico da Figura 7.22, todos os utilizadores consideraram que a introdução do Diagrama de *Features* seria útil, o que faz deste resultado bastante positivo.

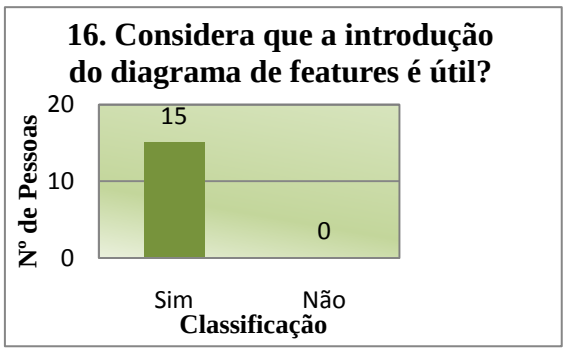

Figura 7.22. Utilidade do Diagrama de *Features*

As 3 questões seguintes – da 17 à 19 – tinham como objectivo obter alguma sugestão para alterar os diagramas produzidos, caso algum utilizador tivesse notado algo que estava errado ou que poderia ser melhorado. No entanto, nenhum utilizador encontrou alguma coisa para sugerir.

17. Faria alguma alteração ao diagrama de features?

18. Faria alguma alteração ao diagrama de classes?

19. Faria alguma alteração ao diagrama de sequência?

20. Considera que a utilização de cenários aspectuais nos diagramas de sequência facilita a compreensão da aplicação dos padrões?

A questão 20 pretendia verificar se a ideia de introduzir mecanismos de Desenvolvimento de Software Orientado a Aspectos vinha, de facto, trazer algumas vantagens na compreensão da aplicação do padrão. Como podemos ver pelo gráfico da Figura 7.23, o resultado desta questão foi muito positivo, pois todos os utilizadores consideraram que "sim", que a utilização dos cenários aspectuais torna mais fácil a compreensão da aplicação do padrão.

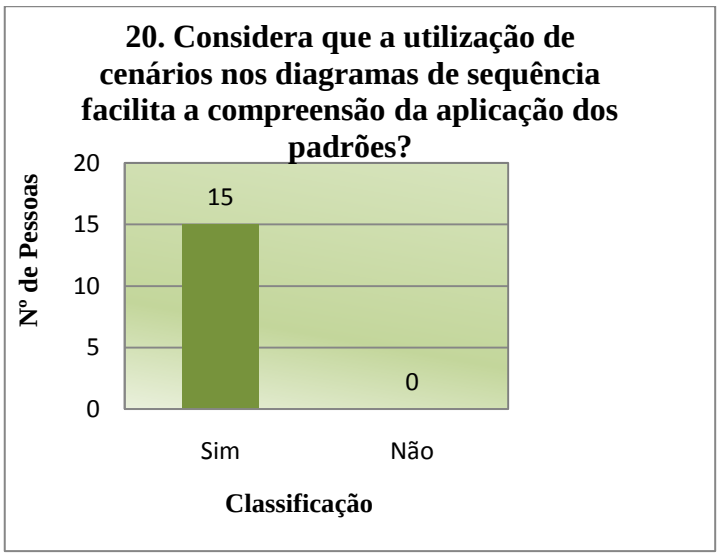

Figura 7.23. Vantagem da Utilização de Cenários Aspectuais

21. Considera que a utilização de cenários aspectuais nos diagramas de sequência ajuda na modularização?

Esta questão pretendia corroborar a ideia de que a utilização de mecanismos de DSOA traz grandes benefícios para a descrição do padrão. Neste caso pretendia-se verificar se estes mecanismos tornam a modularização mais fácil. Pelo gráfico apresentado na Figura 7.24, verificamos que 14 dos 15 utilizadores consideram que "sim", e apenas 1 deles considera que "não". Este resultado pode ser considerado positivo, pois o utilizador que considerou que não seria benéfico para a modularização não tinha conhecimentos sobre os mecanismos DSOA, nem sobre modularização.

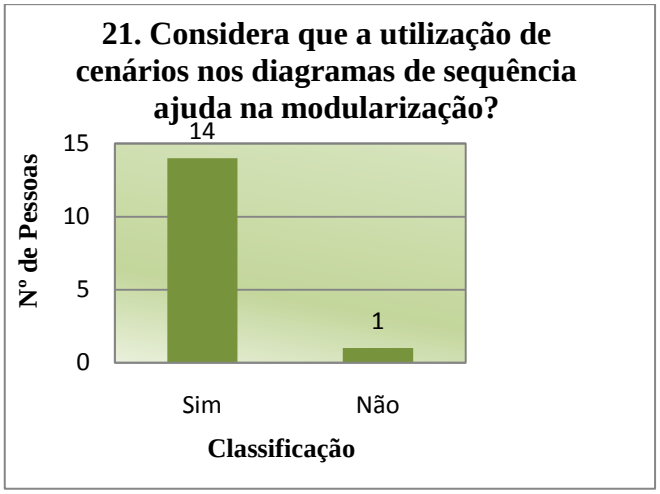

Figura 7.24. Benefício da Utilização de Cenários Aspectuais para a Modularização

22. Como classifica a utilidade dos exemplos apresentados?

Esta questão tinha como objectivo perceber se os utilizadores tinham compreendido os exemplos apresentados e se estes eram explícitos o suficiente para ajudar na compreensão do padrão. Como vemos pelo gráfico da Figura 7.25, apenas dois utilizadores consideraram que a utilidade dos exemplos apresentados era "moderada", sendo que 8 consideraram-nos "úteis" e os restantes 5 consideraram-nos "muito úteis". Desta forma, conclui-se que os exemplos estão bastante claros e conseguem mostrar aos utilizadores como o padrão será aplicado.

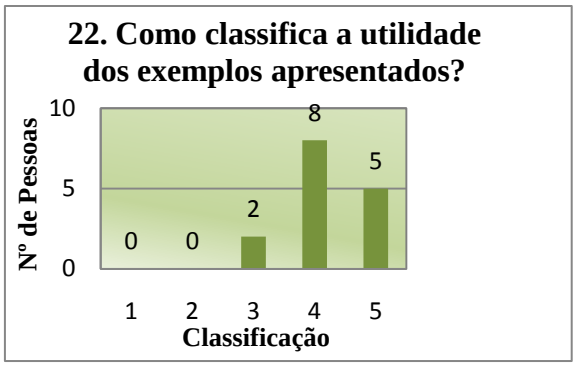

Figura 7.25. Utilidade dos Exemplos Apresentados

23. Como classifica a clareza dos exemplos apresentados?

Esta questão pretende verificar se os utilizadores consideraram os exemplos apresentados claros, ou seja, se tiveram facilidade em compreendê-los. Pelo resultado apresentado no gráfico da Figura 7.26, vê-se que os utilizadores, de um modo geral, acharam os exemplos claros, pois 9 dos 15 utilizadores responderam que os achavam "muito claros", 5 responderam achá-los "claros" e apenas 1 utilizador achou que estavam "moderadamente claros".

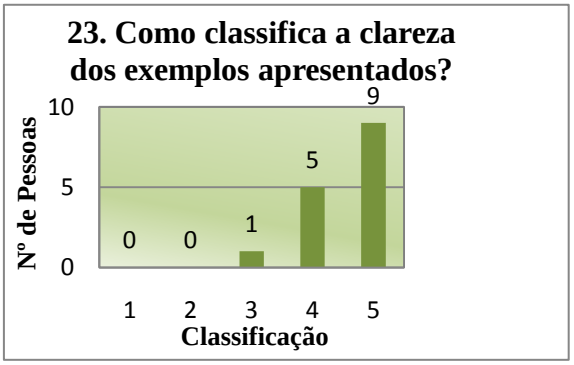

Figura 7.26. Clareza dos Exemplos

As questões que se seguem, 24 a 30, à semelhança das questões 5 a 15, têm o objectivo de obter da parte dos utilizadores sugestões relativamente à descrição de alguns campos do padrão, de forma a poder melhorar a descrição produzida e a perceber que tipo de informação se torna mais ou menos clara e perceptível para os utilizadores. Nas questões 24, 25, 27, 29 e 30, a totalidade dos utilizadores responderam "Não", portanto não acham que seja necessário alterar, acrescentar ou remover nenhum exemplo, padrão relacionado, consequência ou outro aspecto não mencionado na descrição do padrão proposto.

24. Faria alguma alteração aos exemplos?

25. Acrescentaria algum exemplo?

26. Como classifica a pertinência dos padrões relacionados apresentados?

Nesta questão pretendia avaliar-se a pertinência dos padrões relacionados propostos. Apesar dos utilizadores responderem que não alterariam nada neste campo, quando questionados sobre a pertinência dos mesmos – Figura 7.27 – dois dos utilizadores consideram que a pertinência é moderada, o que pode significar que consideraram algum destes padrões não totalmente adequado.

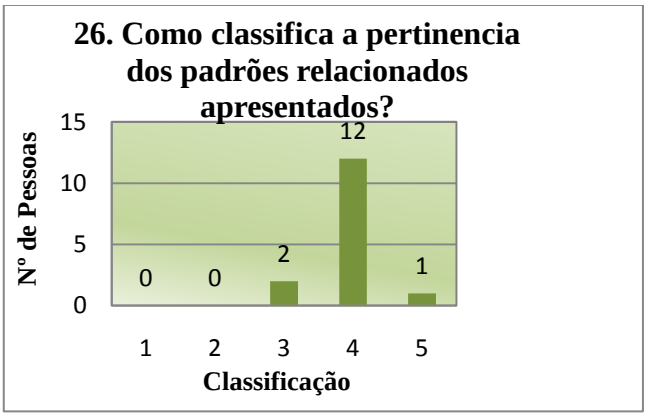

Figura 7.27. Pertinência dos Padrões Relacionados

27. Acrescentava/Removia algum destes padrões?

28. Relativamente às consequências da aplicação do padrão, como as classifica relativamente à sua pertinência?

Nesta questão pretendia-se verificar se os utilizadores consideravam as consequências da aplicação do padrão válidas, e como podemos ver no gráfico da Figura 7.28, 4 dos 15 utilizadores consideram a pertinência "moderada" apesar de não sugerirem que se acrescente ou remova nenhuma consequência.

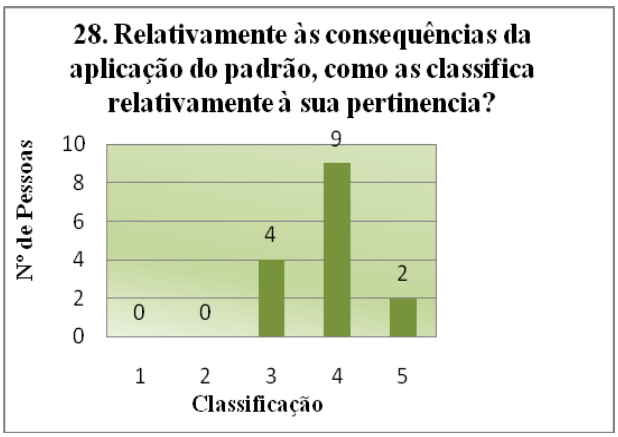

Figura 7.28. Pertinência das consequências da aplicação do Padrão

29. Acrescentava/Removia alguma consequência da aplicação deste padrão?

30. Acrescentava/Alterava mais alguma coisa na definição do Padrão?

Nas duas questões seguintes – 31 e 32 – pretendia-se obter, por parte dos utilizadores, o que estes consideraram ser melhor na descrição e o que consideraram ser pior, de forma a poder alterar a descrição de acordo com as preferências dos utilizadores. Mostram-se abaixo as citações dos pontos positivos e negativos apontados pelos utilizadores:

31. O que considera mais positivo nesta descrição?

- "Obter uma forma generalizável para um problema recorrente – a manutenção de posicionamento de entidade"
- "Penso que a introdução da componente de georreferenciação em determinadas aplicações, *web* ou não, é sempre positiva. A forma simples e organizada como este padrão foi feito, facilita a introdução do mesmo na aplicação, o que facilita muito a sua implementação".

32. E o que considera menos positivo?

- "O nome dado ao padrão é demasiado extenso".

33. Como classifica a facilidade de reutilização deste padrão?

Esta questão foi feita para avaliar se os utilizadores consideraram que a descrição do padrão está suficientemente clara e objectiva para ser fácil utilizá-la. Como vemos pelo gráfico da Figura 7.29, o resultado desta questão é bastante positivo, pois apenas 4 utilizadores consideram a facilidade de reutilização "moderada", sendo que 10 dos utilizadores consideram "fácil" de reutilizar e 1 considerou mesmo "muito fácil".

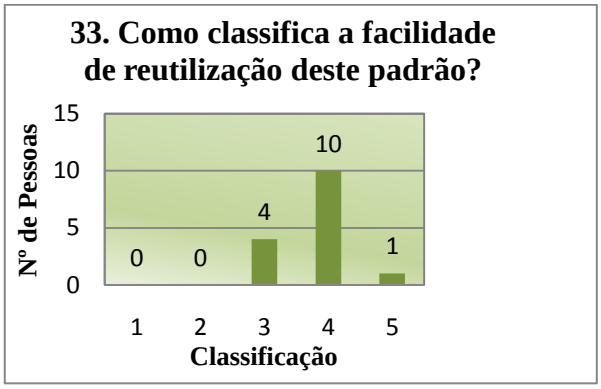

Figura 7.29. Facilidade de Reutilização do Padrão

34. Utilizaria este padrão?

Esta questão pretende avaliar se os utilizadores se sentem realmente convencidos e seguros para usar este padrão. Como podemos ver abaixo, no gráfico da Figura 7.30, apenas 1 utilizador não se sentiu confortável em vir a usar este padrão, os restantes 14 consideraram que se tivessem oportunidade o usariam. Este resultado é bastante positivo, uma vez que o único utilizador a considerar que não utilizaria não era *expert* na área dos SIG, mas sim da área das LDEs. Isto demonstra que os *experts* de SIG consideraram este trabalho importante, tendo assim percebido a sua relevância e a sua utilidade.

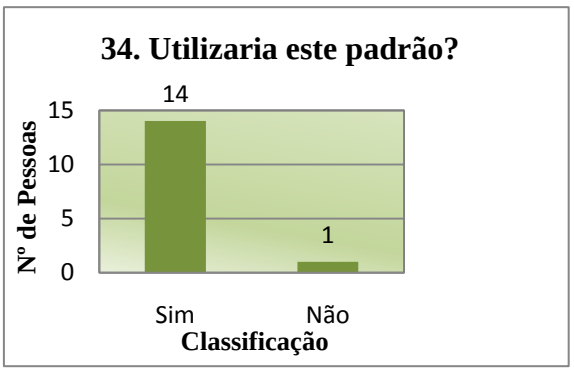

Figura 7.30. Utilização do Padrão

35. Sugestões/Comentários.

Esta questão foi incluída no questionário para que os utilizadores pudessem dizer algo que tivessem notado ao longo do teste que não lhes tivesse sido perguntado, ou fazer alguma sugestão que não tivessem tido oportunidade de fazer até aqui. Nenhum utilizador sentiu necessidade de usar esta questão.

7.3. Ameaças à Avaliação

Quando se faz avaliação de determinado trabalho há que ter em conta que existem factores que podem por em causa a validade dos dados. Algumas destas ameaças são, por exemplo, a falta de avaliação no mundo industrial/empresarial, que daria uma outra perspectiva à avaliação, tanto da ferramenta como da descrição do padrão. No entanto, os utilizadores que testaram a ferramenta e a descrição do padrão têm a vantagem de estar em contacto com tecnologias mais actualizadas, o que nem sempre acontece no mundo empresarial.

A avaliação também foi realizada apenas com 15 utilizadores, o que na verdade constitui um número reduzido. No entanto, apesar de não ter significado estatístico, os resultados obtidos são indicadores da aceitação da solução avaliada.

7.4. Sumário

Ao longo deste capítulo começámos por apresentar os utilizadores que testaram a ferramenta e a descrição do padrão proposta. De seguida, apresentámos uma breve descrição das questões presentes nos 2 questionários efectuados e para algumas questões fizemos uma pequena análise dos resultados, de forma a mostrar qual o *feedback* dos utilizadores relativamente à ferramenta *PatternTool* e à descrição proposta para o padrão testado. Finalmente,

apresentamos alguns factores que podem constituir uma ameaça para a validade dos resultados obtidos neste processo de avaliação.

Com esta avaliação pudemos verificar que de um modo geral os utilizadores consideraram a ferramenta proposta bastante útil, apesar de ter algumas falhas que podem ser resolvidas no futuro. Os utilizadores consideraram a descrição do padrão proposto uma mais-valia para a reutilização dos SIG. Desta forma, podemos dizer que o objectivo desta dissertação terá sido cumprido.

8. Conclusões e Trabalho Futuro

8.1. Conclusões

Nesta dissertação, começou-se por esclarecer os conceitos teóricos relacionados com os Sistemas de Informação Geográfica, o seu funcionamento, objectivos e desenvolvimento ao longo do tempo, nomeadamente os Sistemas de Informação Geográficos na *Internet*. Foi também objectivo a clarificação dos conceitos relacionados com a Engenharia de Requisitos e a Reutilização de Requisitos.

Como os *Web* SIG são sistemas que verificam uma grande volatilidade dos seus requisitos, pois estes podem sofrer constantes alterações, foi também objectivo deste trabalho, o estudo das alterações nos requisitos de um sistema e a identificação dos requisitos voláteis que entretanto se repetem em várias aplicações. Com vista à possibilidade de reutilização destes requisitos, uma vez que o seu comportamento é recorrente em vários sistemas, foram estudados alguns conceitos relacionados com a Análise de Domínio e os Padrões de Análise. Foi criado um diagrama de *features* para a inserção de propriedades geoespaciais em objectos que não as tinham, apresentado na secção 3.2.2.3. Posto isto, foram identificados alguns padrões de análise e foram seleccionados dois deles para serem descritos – estas descrições foram apresentadas no Capítulo 5, nas secções 5.1.1 e 5.1.2 – com base no *template* seleccionado – apresentado também no Capítulo 5, na secção 5.1.

A descrição destes padrões é uma das principais contribuições desta dissertação, na medida em que estes surgem para permitir a reutilização, sendo as suas principais preocupações os modelos conceptuais, a flexibilidade e a capacidade de reutilização dos módulos de sistemas que resultam da aplicação destes padrões. Outra importante contribuição desta dissertação foi a inserção do diagrama de *features* na descrição do padrão, de forma a permitir que se tenha um pleno conhecimento do domínio e variabilidade dos conceitos em que o padrão se enquadra, para que seja possível uma decisão consciente, quando for altura de decidir sobre reutilizar ou não o padrão em causa. Finalmente, a introdução de mecanismos de Desenvolvimento de *Software* Orientado a Aspectos, na modelação comportamental do

padrão, foi outra das principais contribuições desta dissertação. Mais concretamente aplicaram-se cenários aspectuais e composição. Estes mecanismos DSOA têm as vantagens de minimizar a replicação de código, melhorando a coesão entre os diversos módulos; de reduzir o entrelaço do sistema e, consequentemente, de aumentar a possibilidade de reutilização; e de facilitar a evolução no desenvolvimento de sistemas de *software* complexos.

Foi ainda desenvolvida uma aplicação que tem como objectivo servir de suporte computacional para a criação dos padrões – Ferramenta *PatternTool*, apresentada no capítulo 6. Esta ferramenta, além do suporte computacional à descrição de padrões, tem a vantagem de agregar a descrição do padrão, a edição de diagramas de *Features*, de diagramas de classes e de diagramas de sequência, tudo na mesma aplicação.

Para esta dissertação, foi usado como o caso de estudo a adição de georreferenciação no sistema CLIP. Este sistema faz a gestão de uma base de dados de todos os alunos e docentes da Universidade Nova de Lisboa, bem como dos edifícios e salas dos campus das faculdades, onde decorrem as aulas. Pretendeu-se avaliar o impacto, na aplicação, de adicionar, por exemplo, a capacidade de saber onde determinada aula está a decorrer em determinado momento, não apenas através das indicações da hora, do edifício, e da identificação da sala, mas a sua localização visual num mapa. Poderá também adicionar-se a capacidade de localização de uma determinada pessoa num determinado momento, tendo em conta as aulas que dá ou assiste e o local onde trabalha/estuda, dentro do campus.

8.2. Limitações

A ferramenta desenvolvida nesta dissertação tem algumas limitações que em alguns casos se prendem com as próprias limitações da *framework* usada para a desenvolver. Nomeadamente, quando usamos os sub-editores, colocamos os elementos em determinadas posições e com determinada disposição. No entanto, quando gravamos, para que os objectos fiquem visíveis no editor principal o que acontece é que estes são colocados todos sensivelmente na mesma posição sobrepostos, pois o Eclipse não guarda as posições dos elementos de forma a poder recolocá-los com a mesma disposição.

É também de notar que no diagrama de *Features* a forma como as relações de *Or, OrOpcional, Alternativa e AlternativaOpcional* estão representadas não é a mais intuitiva, pois é necessário fazer a ligação opcional ou obrigatória entre as *features* e depois entre as ligações criar a relação.

É ainda importante ter em conta que devido ao tempo reduzido para a realização de uma dissertação de mestrado, no que diz respeito à avaliação faltou a concretização de um processo de avaliação no mundo industrial/empresarial, que daria uma outra perspectiva à avaliação, tanto da ferramenta como da descrição do padrão.

Relativamente à descrição do padrão poderá haver algumas limitações se um *expert* da área dos SIG não tiver qualquer conhecimento de DSOA. Nestes casos poderá ser complicado para estes compreenderem a descrição dos padrões propostos.

8.3. Trabalho Futuro

Para dar seguimento ao trabalho realizado nesta dissertação, propõe-se, como trabalho futuro, a identificação de outros padrões de análise existentes nas aplicações de Sistemas de Informação Geográfica para a *Web*, bem como a descrição de outros padrões já identificados. Estes incluem, por exemplo, a adição de restrições temporais aos objectos (e.g. tornar uma via disponível de acordo com um horário); a adição perfis de utilizador (e.g. seleccionar determinado tipo de via de acordo com o perfil do utilizador, como no caso de uma pessoa com deficiências motoras, em que se deveria seleccionar para ela, num determinado caminho, apenas vias adequadas).

Relativamente à ferramenta, seria interessante criar um novo editor principal que incluísse uma tabela onde estaria o nome de todos os padrões adicionados o que daria acesso à descrição de cada padrão – desenvolvida nesta dissertação. Poder-se-ia, neste contexto, editar os padrões já criados e criar novos padrões, ou apenas consultá-los. Desta forma, a ferramenta funcionaria como um catálogo de padrões de análise. Seria também importante fazer algumas alterações no editor do Diagrama de Sequência, recorrendo, provavelmente à alteração do código java gerado, de forma a obter um diagrama mais perfeito. Também seria interessante fazer algumas alterações no editor do Diagrama de *Features* no sentido de modificar a representação do *Or*, *OrOpcional*, *Alternativa* e *AlternativaOpcional*. Finalmente, é importante que se criem algumas restrições usando OCL, como foi sugerido por alguns utilizadores, de forma a impedir por exemplo, que no Diagrama de *Features* se possam ligar *features* duas vezes entre elas, ou fazer ciclos.

9. Bibliografia

Arango, G., *Domain Analysis: From Art Form To Engineering Discipline*, Proceedings of the 5th International Workshop on Software Specification and Design, Pittsburgh, EUA, págs. 152-159, Maio de 1989.

Araújo, J., Whittle, J., Kim, D. *Modeling and Composing Scenario-Based Requirements with Aspects*, The 12th IEEE International Requirements Engineering Conference (RE2004), Quioto, Japão, IEEE Computer Society, Setembro de 2004.

Batory, D., Johnson, C., MacDonald, B., Heeder, D, *Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study*, ACM Transactions on Software Engineering and Methodology, Vol. 11, Nº. 2, págs. 191–214, Abril de 2002.

Borges, K., Davis Jr., C., Laender, A., *OMT-G: An Object-Oriented Data Model for Geographic Applications*, GeoInformática, Vol. 5:3, págs. 221-260, Setembro de 2001.

Chow, E, *The Potential of Maps APIs for Internet GIS Applications*, Transactions in GIS, Michigan, Vol. 12:2, págs. 179-191, 2008.

Coad, P., North, D., Mayfield, M., *Object models-strategies, patterns and applications*, Prentice-Hall, *New Jersey*, 1995.

Dana, P., *Geographic Information System Loran-C coverage Modeling*, 22nd Annual Technical Symposium, Santa Barbara, California, págs. 279-286, 1993.

Deursen, A., Klint, P., *Domain-Specific Language Design Requires Feature Descriptions*, Journal of Computing and Information Technology, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, N.º 1, págs. 1–17, 2002.

Dragicevic, S., *The potential of Web-based GIS*, Journal of Geographic Systems, Springer-Verlag, Vol. 6, págs. 79-81, 2004.

Emfatic, http://wiki.eclipse.org/Emfatic, 2008.

EuGENia, http://www.eclipse.org/gmt/epsilon/doc/articles/eugenia-gmftutorial, 2008.

ESRI - Environmental Systems Research Institute, Inc. Consultado via página *Web* pela última vez em 4 de Janeiro de 2010, http://www.esri.com/library/bestpractices/gis-and-science.pdf, Novembro de 2008.

Figueiredo, E., Garcia, A., Sant'Anna, C., Kulesza, U., Lucena, C., *Assessing Aspect-Oriented Artifacts: Towards a Tool-Supported Quantitative Method*, 9th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'05), Glasgow, Reino Unido, 25 de Julho de 2005.

Fu, Y., Ding, J., Bording, P., *An Approach for Modeling and Analyzing Crosscutting Concerns*, The 5th IEEE International Conference on Services Operations, Logistics and Informatics, Chicago, EUA, 22 a 24 de Julho de 2009.

GIS – Geographic Information System. Consultado via página *Web* pela última vez em 1 de Fevereiro de 2010, http://www.gis.com/content/what-can-you-do-gis.

Gordillo, S., Balaguer, F., Mostaccio, C., Neves, F., *Developing GIS Applications with Objects: A design Patterns Approach*, GeoInformática, Vol. 3:1, págs. 7-32, 1999.

Hamza, H., Fayad, M., *Appying Analysis PatternsThrough Analogy: Problems and Solutions*, Journal of Object Technology, Vol. 3:4, págs. 197-208, 2004.

Harel, D., Rumpe, B., *Modeling Languages: Syntax, Semantics and All That Stuff – Part I: The Basic Stuff*, Technical Report MCS00-16, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, Israel, 2000.

Harrer, A., Devedzic, V., *Design and Analysis Patterns in ITS Architectures*, Proceedings of the International Conferecen on Computers in Education (ICCE'02), Auckland, Nova Zelândia, 3 a 6 de Dezembro de 2002.

Henderson-Sellers, B., France, R., Georg, G., Reddy, R., *A method engineering approach to developing aspect-oriented modelling processes based on the open process framework*, Information and Software Technology, Vol. 1.49, N.º 7, págs. 761-773, 2007.

Hudak, P., *Modular domain specific languages and tools*, in Proceedings of the 5th International Conference on Software Reuse(JCSR'98), P. Devanbu and J. Poulin, Eds. IEEE Computer Society, págs. 134 – 142, 1998.

Jing-zhong, W., Hui-dan, L., *Research on the Web GIS Technology Based on MapXtreme*, International Conference on Computer Science and Software Engineering 2008 (CSSE 2008), Wuhan, China, Vol. 2, págs. 123-126, 12 a 14 Dezembro de 2008.

Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A., *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Domain Analysis Project, Software Engineering Institute, Carnegie Mellon University Pittsburgh, Pennsylvania, Novembro de 1990.

Kraak, M., *The role of the map in a Web-GIS environment*, Journal Geographical Systems, Vol. 6, págs. 83-93, 2004.

Konrad, S., Cheng, B., Campbell, L., *Object Analysis Patterns for Embedded Systems*, IEEE Transactions on Software Engineering, Vol. 30:12, págs. 970-992, Dezembro de 2004.

Konrad, S., Cheng, B., *Requirements Patterns for Embedded Systems*, Proc. Of IEEE Joint International Conference on Requirements Engineering, Essen, Alemanha, Setembro de 2002.

Kotonya, G., Sommerville, I., *Requirements Engineering – Processes and Techniques*, Wiley, 2004.

Kou-gen, Z., Rahim, S., Yun-he, P., *Web GIS: Implementation Issues*, Chinese Geographical Science, Beijing, China, Vol. 10:1, págs. 74-79, 2000.

Lisboa Filho, J., Iochpe, C., Beard, K., *Applying Analysis Patterns in the GIS Domain*, Proc. 10th Annual Colloquium of the SIRC, Dunedin, Nova Zelândia, 1998.

Luqun, L., Minglu, L., *A Research on Development of mobile GIS architecture*, International Society for Environmental Information Sciences (ISEIS'2004), Regina, Canada, 2004.

Marshall, J., *Developing Internet-Based GIS Applications*, ERSI International User Conference, INDUS Corporation, 2000.

Medeiros, C., Pires, F., *Databases for GIS*, ACM SIGMOD Record, Vol. 23, N.º 1, págs. 107-115, 1994.

Mei, H., Zhang, W., Gu, F., *A Feature Oriented Approach to Modeling and Reusing Requirements of Software Product Lines*, Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03), IEEE Computer Science, Texas, EUA, 3 a 6 de Novembro de 2003.

Mernik, M., Heering, J., Sloane, A., *When and How to Develop Domain-Specific Languages*, ACM Computing Surveys, Vol. 37, N.º 4, págs. 316–344, Dezembro de 2005.

Mezei, G., Lengyel, L., Levendovszky, T., Charaf, H., *A model Transformation for Automated Concrete Syntax Definitions of Metamodeled Visual Languages*, In Proc. of 2nd International Workshop on Graph and Model Transformation 2006 (GraMot'06), Electronic Communications of the EASST, Vol. 4, Brighton, Reino Unido, 8 de Setembro de 2006.

Moreira, A., J. Araújo, J. Whittle, *Modeling Volatile Concerns as Aspects*, *18th Conference on Advanced Information Systems Engineering* (CAiSE 2006), Luxemburgo, Lecture Notes in Computer Science, Springer, 5 a 9 de Junho de 2006.

Neighbors. J., *Software Construction Using Components*, Tese de Doutoramento no Departamento de Informática e Ciências de Computador da Universidade da California, Irvine, 1961.

Oliveira, A., *Modelação de Aplicações SIG com Aspectos*, Dissertação de Mestrado em Engenharia Informática, Departamento de Informática, Universidade Nova de Lisboa Faculdade de Ciências e Tecnologia, 2009.

Özgür, T., *Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the context of the Model Driven Development*, School of Engineering. Ronneby, Suécia, 2007.

Pantoquilho, M., Raminhos, R., Araújo, J., *Analysis Patterns Specifications: Filling the Gaps*, Viking PLoP 2003, Bergen, Noruega, 2003.

Pejic, A., Pletl, S., Pejic, B., *An expert system for tourists using Google Maps API*, SISY '09. 7th International Symposium on Intelligent Systems and Informatics 2009, págs. 317-322, 25 e 26 de Setembro de 2009.

Pressman, R., *Software Engineering: A Practitioner's Approach*, 6th Edition, McGraw Hill, 2005.

Prieto-Díaz, R., *Domain Analysis: An Introduction*, *ACM SIGSOFT Software Engineering Notes*, Vol. 15:2, págs. 47-54, 1990.

Ramzan, S., Ikram, N., *Making Decision in Requirement Change Management*, Information and Communication Technologies, *ICICT2005*, Karachi, Paquistão, 27 e 28 de Agosto de 2005.

Ramzan, S., Ikram, N., Requirement Change Management Process Models: Activities, Artifacts and Roles, INMIC06, Proceedings of The 10th IEEE International Multi-topic Conference, págs. 219–223, Islamabad, Paquistão, 23 e 24 de Dezembro de 2006.

Rashid, A., Moreira, A., Araújo, J., *Modularization and Composition of Aspectual Requirements*, Intl. Conf. on AOSD, Boston, EUA, ACM Press, Março de 2003.

Sommerville, I., *Software Engineering 8*, Harlow, Inglaterra, Addison-Wesley, 2007.

Sommerville, I., Sawyer, P., *Requirements Engineering – A Good Practice Guide*, John Wiley, 1997.

Steinberg, D., Budinsky, F., Paternostro, M., Merks, E., *EMF: Eclipse Modeling Framework*, Addison-Wesley Professional, 2ª Edição , 16 de Dezembro de 2008.

Sutton Jr, S., Rouvellou, I., *Concern Modeling for Aspect-Oriented Software Development*, In Aspect-Oriented Software Development, Addison Wesley, págs. 479 - 506, 2005.

Thibault, S. A., *Domain-specific languages: Conception, implementation and application*, Tese de Doutoramento, Universidade de Rennes, 1998.

Urbieta, M., Rossi, G., Gordillo, S., Rodrigues, A., Araujo, J., Moreira, A., *An Aspect-Oriented Approach for Spatial Concerns in Web-GIS Applications*, Submetido para a revista GeoInformática, 2010.

Whittle, J., Jayaraman, P., *Mata: A Tool for Aspect-Oriented Modeling based on Graph Transformations*, Workshop on Aspect-Oriented Modeling at MODELS 07, Nashville, EUA, 2007.

Zipf, A., Merdes, M., *Is aspect-orientation a new paradigm for GIS development? On the Relationship of Geoobjects, Aspects, and Ontologies*, 6th Agile Conference on Geographic Information Science, Lyon, França, págs. 717-728, 24 a 26 de Abril de 2003.

10. Anexo A – Questionário 1: Avaliação da Ferramenta

Linguagem

L1. Com que facilidade aprendeu os conceitos?

Muito Difícil 1☐ 2☐ 3☐ 4☐ 5☐ Muito Fácil

L2. Com que facilidade identificou os símbolos que representavam os conceitos?

Muito Difícil 1☐ 2☐ 3☐ 4☐ 5☐ Muito Fácil

Quais considera inadequados?

L3. Com que facilidade identificou o texto que representava os conceitos?

Muito Difícil 1☐ 2☐ 3☐ 4☐ 5☐ Muito Fácil

Quais considerou inadequados?

L4. Com que frequência cometeu erros devido à semelhança dos símbolos?

Muito Frequente 1☐ 2☐ 3☐ 4☐ 5☐ Pouco Frequente

L5. Com que frequência cometeu erros devido à ambiguidade do vocabulário?

Muito Frequente 1☐ 2☐ 3☐ 4☐ 5☐ Pouco Frequente

L6. Com que frequência se sentiu incapaz de expressar o que desejava?

Muito Frequente 1☐ 2☐ 3☐ 4☐ 5☐ Pouco Frequente

Caso se aplique, dê o exemplo de um caso.

Elaboração do Cenário

C1. Com que frequência sentiu necessidade de consultar a documentação durante a execução do cenário?

Muito Frequente 1☐ 2☐ 3☐ 4☐ 5☐ Pouco Frequente

C2. Com que frequência teve necessidade de questionar o supervisor durante a execução do cenário?

Muito Frequente 1☐ 2☐ 3☐ 4☐ 5☐ Pouco Frequente

C3. Como classifica o seu grau de confiança durante a execução do cenário dado?

Pouco Confiante 1☐ 2☐ 3☐ 4☐ 5☐ Muito Confiante

C4. Com que frequência se sentiu confuso durante a execução do cenário?

Muito Frequente 1☐ 2☐ 3☐ 4☐ 5☐ Pouco Frequente

C5. Como classifica a dificuldade do cenário?

Muito Difícil 1☐ 2☐ 3☐ 4☐ 5☐ Muito Fácil

C6. O que considerou mais difícil de fazer?

C7. Como se sente relativamente à correcção do cenário elaborado?

Pouco Confiante 1☐ 2☐ 3☐ 4☐ 5☐ Muito Confiante

Usabilidade

Tempo gasto a elaborar o template do Padrão fornecido. ___minutos

F1. Qual a sua impressão geral da ferramenta?

Muito Má 1☐ 2☐ 3☐ 4☐ 5☐ Muito Boa

F2. Como se sentiu ao fazer alterações?

Pouco Confiante 1☐ 2☐ 3☐ 4☐ 5☐ Muito Confiante

F3. Considera que foi fácil passar o padrão do papel para a ferramenta?

Muito Difícil 1☐ 2☐ 3☐ 4☐ 5☐ Muito Fácil

F4. O resultado final do padrão, na ferramenta, era o que estava à espera?

Sim☐ Não☐

F5. Que alterações propunha para a ferramenta?

F6. Compreendeu o processo de criação do Padrão?

Sim☐ Não☐

Se respondeu **não**, o que faltou?

F7. Com que facilidade executou o processo em F6?

Muito Difícil 1☐ 2☐ 3☐ 4☐ 5☐ Muito Fácil

F8. Teve dificuldades a utilizar a ferramenta?

Sim☐ Não☐

F9. Comparando a descrição do padrão que lhe foi fornecida em papel e o gerado na ferramenta, como classifica o resultado esperado?

Muito Diferente 1☐ 2☐ 3☐ 4☐ 5☐ Muito Semelhante

F10. Ao mudar de um sub-editor (Diagrama de Classes, de Features ou de Sequência) para o editor principal (Template), denotou alguma perda de informação?

Sim☐ Não☐

Se a resposta foi **sim,** descreva os casos.

F11. Como classifica a ferramenta comparando com outras ferramentas com que já trabalhou?

Muito Pior 1☐ 2☐ 3☐ 4☐ 5☐ Muito Melhor

F12. Considera a ferramenta útil?

Sim☐ Não☐

F13. O que acha que a descrição obtida na ferramenta traz de melhor em relação à que lhe foi fornecida em papel?

F14. E o que acha que trás pior?

F15. Qual a maior dificuldade que encontrou no uso da ferramenta?

F16. Quais são, na sua opinião, os pontos fracos desta ferramenta?

F17. E quais pontos fortes da ferramenta?

F18. O que sugere para que a ferramenta possa ser melhorada?

11. Anexo B – Questionário 2: Avaliação do Padrão

1. Como classifica a relevância do padrão proposto?

Pouco Relevante 1☐ 2☐ 3☐ 4☐ 5☐ Muito Relevante

2. Acha que este pode ser reutilizado em várias aplicações de diversas áreas?

Sim☐ Não☐

3. Acha que as aplicações SIG podem beneficiar deste padrão?

Sim☐ Não☐

4. Como classifica a clareza e simplicidade da definição do padrão?

Muito Más 1☐ 2☐ 3☐ 4☐ 5☐ Muito Boas

5. Como classifica a escolha do nome do Padrão?

Muito Mau 1☐ 2☐ 3☐ 4☐ 5☐ Muito Bom

6. Tem alguma sugestão melhor para o nome do Padrão?

Sim☐ Não☐

Se respondeu **sim**, qual?

7. Acha que a introdução da lista de eventos é importante para quem vier a reutilizar este padrão?

Sim ☐ Não ☐

8. Como classifica a pertinência dos eventos apontados na lista do padrão?

Pouco Pertinente 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Pertinente

9. Acrescentaria ou removeria algum evento?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

10. Como classifica a pertinência dos requisitos funcionais descritos no padrão?

Pouco Pertinente 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Pertinente

11. Acrescentaria ou removeria algum requisito funcional?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

12. Como classifica a pertinência dos requisitos não-funcionais descritos no padrão?

Pouco Pertinente 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Pertinente

13. Acrescentaria ou removeria algum requisito não-funcional?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

14. Como classifica a pertinência das dependências entre os requisitos do padrão?

Pouco Pertinente 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Pertinente

15. Alteraria alguma dependência?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

16. Considera que a introdução do diagrama de features é útil?

Sim ☐ Não ☐

17. Faria alguma alteração ao diagrama de features?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

18. Faria alguma alteração ao diagrama de classes?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

19. Faria alguma alteração ao diagrama de sequência?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

20. Considera que a utilização de cenários nos diagramas de sequência facilita a compreensão da aplicação dos padrões?

Sim ☐ Não ☐

21. Considera que a utilização de cenários nos diagramas de sequência ajuda na modularização?

Sim ☐ Não ☐

22. Como classifica a utilidade dos exemplos apresentados?

Pouco Útil 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Útil

23. Como classifica a clareza dos exemplos apresentados?

Muito Más 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Boas

24. Faria alguma alteração aos exemplos?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

25. Acrescentaria algum exemplo?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

26. Como classifica a pertinência dos padrões relacionados apresentados?

Pouco Pertinente 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Pertinente

27. Acrescentava/Removia algum destes padrões?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

28. Relativamente às consequências da aplicação do padrão, como as classifica relativamente à sua pertinência?

Pouco Pertinente 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Pertinente

29. Acrescentava/Removia alguma consequência da aplicação deste padrão?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

30. Acrescentava/Alterava mais alguma coisa na definição do Padrão?

Sim ☐ Não ☐

Se respondeu **sim**, qual?

31. O que considera mais positivo nesta descrição?

32. E o que considera menos positivo?

33. Como classifica a facilidade de reutilização deste padrão?

Muito Difícil 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ Muito Fácil

34. Utilizaria este padrão?

Sim ☐ Não ☐

Se respondeu **não**, porquê?

35. Sugestões/Comentários.

12. Anexo C – Metamodelo *emf* do Editor Principal

```
@namespace(uri="padrao", prefix="padrao")
package padrao;

@gmf.diagram(foo="bar")
class Padrao {
  val PadraoTemplate[1] Padrao;
}

@gmf.node(figure="rectangle", label="nomePadrao", label.pattern="Pattern
Template", tool.name="Pattern Template")
class PadraoTemplate {
  attr String nomePadrao;

  @gmf.compartment(layout="list")
  val NomePadrao[1] nomePad;

  @gmf.compartment(layout="list")
  val ProblemaPadrao[1] problemaPad;

  @gmf.compartment(layout="list")
  val ContextoPadrao[1] contextoPad;

  @gmf.compartment(layout="list")
  val ReqsPadrao[1] PadReqs;

  @gmf.compartment(layout="list")
  val ListaEventosPadrao[1] listaEvtsPad;

  @gmf.compartment(layout="list")
  val Modelacao[1] modelacaoPad;

  @gmf.compartment(layout="list")
  val Consequencias[1] consequenciasPad;

  @gmf.compartment(layout="list")
  val Exemplos[+] exemplosPad;
  @gmf.compartment(layout="list")
  val PadroesRelacionados[1] PadroesRelacPad;
}

@gmf.node(figure="rectangle", label="nomeP", label.pattern="Name: {0}",
tool.name="Pattern's Name", tool.description="Add the Pattern's Name")
class NomePadrao {
  attr String nomeP;
}

@gmf.node(figure="rectangle", label="contextoP", label.pattern="Context:
{0}", tool.name="Pattern's Context", tool.description="Add the Pattern's
Context")
```

```
class ContextoPadrao {
  attr String contextoP;
}

@gmf.node(figure="rectangle", label="problemaP", label.pattern="Problem:
{0}", tool.name="Patterns's Problem", tool.description="Add the Patterns's
Problem")
class ProblemaPadrao {
  attr String problemaP;
}

@gmf.node(figure="rectangle", label="listaEventosP", label.pattern="Events
List: {0}", tool.name="Pattern's Events List", tool.description="Add
Pattern's Events List")
class ListaEventosPadrao {
  attr String listaEventosP;

  @gmf.compartment(layout="list")
  val Evento[*] evento;
}

@gmf.node(figure="rectangle", label="ident, nomeEvento",
label.pattern="{0}. {1}", tool.name="Pattern's Event",
tool.description="Add One Pattern's Event")
class Evento {
  attr int ident = 1;
  attr String nomeEvento;
}

@gmf.node(figure="rectangle", label="nomeR", label.pattern="Requirements:
{0}", tool.name="Pattern's Requirements Analysis", tool.description="Add
Pattern's Requirements Analysis")
class ReqsPadrao {
  attr String nomeR;
  val Requisitos[2] ReqPad;

  @gmf.compartment(layout="list")
  val Funcionais[+] f;

  @gmf.compartment(layout="list")
  val NaoFuncionais[+] nf;

  @gmf.compartment(layout="list")
  val Dependencias[1] DepsPad;

  @gmf.compartment(layout="list")
  val Actores[1] ActsPad;
}

@gmf.link(source="LigacaoIn", target="LigacaoOut", style="dash", width="2",
color="255,69,0", target.decoration="arrow", tool.name="Dependency
Connection Between Requirements", tool.description="Add a Dependency
Connection Between Requirements")
class Ligacao {
  ref Requisitos LigacaoIn;
  ref Requisitos[1] LigacaoOut;
}

@gmf.node(label="ident, descricao")
abstract class Requisitos {
  attr String ident = "1";
```

```
  attr String descricao;
  val Ligacao[*] ligacoes;
}

@gmf.node(figure="rounded", label.pattern="FR{0}. {1}", label.icon="false",
tool.name="Functional Requirement", tool.description="Add a Functional
Requirement to the Requirements Section")
class Funcionais extends Requisitos {
}

@gmf.node(figure="ellipse", label.pattern="NFR{0}. {1}",
label.icon="false", tool.name="Non-Functional Requirement",
tool.description="Add a Non-Functional Requirement to the Requirements
Section")
class NaoFuncionais extends Requisitos {
}

@gmf.node(figure="rectangle", label="Id", label.pattern="Dependencies:
{0}", tool.name="Dependencies Between Requirements", tool.description="Add
a Section to Represent the Dependencies Between Requirements")
class Dependencias {

  @gmf.compartment(foo="bar")
  val Requisitos[2..*] DepReq;
  attr String Id;
}

@gmf.node(figure="rectangle", label.pattern="Actors: {0}", label="nome",
tool.name="Actors", tool.description="Add the Actors to the Requirements
Section")
class Actores {
  attr String nome;
}

@gmf.node(figure="rectangle", label="modelacao", label.pattern="Modelation:
{0}", tool.name="Modelation", tool.description="Add a Modelation Section to
the Pattern")
class Modelacao {

  @gmf.compartment(layout="list")
  val Estrutural[1] estruturalPad;
  attr String modelacao;

  @gmf.compartment(layout="list")
  val Comportamental[1] comportamentalPad;
}

@gmf.node(figure="rectangle", label="estrutural", tool.name="Structural
Modelation", tool.description="Add a Structural Modelation Section to the
Pattern")
class Estrutural {
  attr String estrutural;

  @gmf.compartment(foo="bar")
  val DiagFeatures[1] dfeatures;

  @gmf.compartment(foo="bar")
  val DiagClasses[1] dclasses;
}
```

```
@gmf.node(figure="rectangle", label="comportamental", tool.name="Behavioral
Modelation", tool.description="Add a Behavioral Modelation Section to the
Pattern")
class Comportamental {
  attr String comportamental;

  @gmf.compartment(foo="bar")
  val DiagSequencia[1] dsequencia;
}

@gmf.node(figure="rectangle", label="consequenciasP",
label.pattern="Consequences: {0}", tool.name="Pattern's Application
Consequences", tool.description="Add Pattern's Application Consequences to
the Pattern")
class Consequencias {
  attr String consequenciasP;
}

@gmf.node(figure="rectangle", label="exemplosP", label.pattern="Examples:
{0}", tool.name="Examples", tool.description="Add an Examples Section to
the Pattern")
class Exemplos {
  attr String exemplosP;

  @gmf.compartment(layout="list")
  val Funcionais[+] ReqsEx;

  @gmf.compartment(foo="bar")
  val DiagFeatures[1] dFeatureEx;

  @gmf.compartment(foo="bar")
  val DiagClasses[1] dClasseEx;

  @gmf.compartment(foo="bar")
  val DiagSequencia[1..3] dSequenciaEx;
}

@gmf.node(figure="rectangle", label="padroesRelacionadosP",
label.pattern="Related Patterns: {0}", tool.name="Related Patterns
Section", tool.description="Add a Related Patterns Section to the Pattern")
class PadroesRelacionados {
  attr String padroesRelacionadosP;

  @gmf.compartment(layout="list")
  val PadraoRel[*] padRel;
}

@gmf.node(figure="rectangle", label="idPadrao, nomePadraoRel",
label.pattern="{0}. {1}", tool.name="Related Pattern",
tool.description="Add One Related Pattern to the Pattern")
class PadraoRel {
  attr int idPadrao = 1;
  attr String nomePadraoRel;
}

@gmf.node(figure="rectangle", label="diagClasses", tool.name="Class
Diagram", tool.description="Add a Class Diagram to the Pattern. Double
click to open a new editor.")
class DiagClasses {

  @gmf.compartment(foo="bar")
```

```
  val Classe[+] classeDC;
  attr String diagClasses;
}

@gmf.node(figure="rounded", label="classe", tool.name="Class",
tool.description="Add a new Class to the Diagram")
class Classe {

  @gmf.compartment(layout="list")
  val Atributo[+] atribDC;

  @gmf.compartment(layout="list")
  val Metodo[+] metodoDC;
  val Dependencia[+] dependenciaDC;
  val Generalizacao[+] generalizacaoDC;
  attr String classe;
}

@gmf.node(figure="rectangle", label="atrib", tool.name="Attribute",
tool.description="Add a new Attribute to the Class")
@gmf.compartment(layout="list")
class Atributo {
  attr String atrib;

  @gmf.compartment(layout="list")
  val TipoAtributo[1] tipoAtributo;
}

@gmf.node(figure="rectangle", label="tipoAtrib", tool.name="Attribute
Type", tool.description="Add the Attribute Type to the Attribute")
class TipoAtributo {
  attr String tipoAtrib;
}

@gmf.node(figure="rectangle", label="metodo", tool.name="Method",
tool.description="Add a new Method to the Class")
@gmf.compartment(layout="list")
class Metodo {
  attr String metodo;

  @gmf.compartment(layout="list")
  val TipoRetornoMet[1] tipoRetornoMetodo;

  @gmf.compartment(layout="list")
  val ParametrosMetodo[+] parametrosMetodo;
}

@gmf.node(figure="rectangle", label="tipoRetornoMet", tool.name="Method's
Return Type", tool.description="Add the Method's Return Type to the
Method")
class TipoRetornoMet {
  attr String tipoRetornoMet;
}

@gmf.node(figure="rectangle", label="parametrosMet", tool.name="Method's
Parameter", tool.description="Add a Method's Parameter")
class ParametrosMetodo {
  attr String parametrosMet;

  @gmf.compartment(layout="list")
  val TipoParametro[1] tipoParam;
```

```
}

@gmf.node(figure="rectangle", label="tipoParam", tool.name="Parameter's
Type", tool.description="Add a Parameter's Type to the Parameter")
class TipoParametro {
  attr String tipoParam;
}

@gmf.link(label="cardinalidadeIn, cardinalidadeOut", label.pattern="{0}
,           {1}", source="dependenciaDCin", target="dependenciaDCout")
abstract class Dependencia {
  ref Classe[1] dependenciaDCout;
  attr String cardinalidadeIn;
  attr String cardinalidadeOut;
  ref Classe[1] dependenciaDCin;
}

@gmf.link(style="dot", width="2", color="56,114,114",
target.decoration="filledrhomb", tool.name="Aggregation",
tool.description="Add an Aggregation Relation Between two Classes")
class Agregacao extends Dependencia {
}

@gmf.link(style="dot", width="2", color="159,0,159",
target.decoration="arrow", tool.name="Composition", tool.description="Add a
Composition Relation Between two Classes")
class Composicao extends Dependencia {
}

@gmf.link(source="generalizacaoIn", target="generalizacaoOut",
style="solid", width="1", color="0,0,174", target.decoration="closedarrow",
tool.name="Generalization", tool.description="Add a Generalization Relation
Between two Classes")
class Generalizacao {
  ref Classe[1] generalizacaoIn;
  ref Classe[1] generalizacaoOut;
}

@gmf.node(figure="rectangle", label="diagSequencia", tool.name="Sequence
Diagram", tool.description="Add a Sequence Diagram to the Pattern. Double
click to open a new editor.")
class DiagSequencia {
  attr String diagSequencia;

  @gmf.compartment(foo="bar")
  val Fragmentos[+] elems;
}

@gmf.node(label="tipoFrag", tool.name="Fragment", tool.description="Add a
Fragment to the Diagram. These fragments can be of type: Sequence Diagram,
ALT, OPT, LOOP, PAR, BREAK e DivFrag, the last one is used to divide
fragments of type ALT and PAR.")
class Fragmentos {
  attr TipoFrag tipoFrag;

  @gmf.compartment(foo="bar")
  val Condicao condicaoFrag;

  @gmf.compartment(foo="bar")
  val Fragmentos[*] frags;
```

```
  @gmf.compartment(foo="bar")
  val Objectos[*] obj;
  val Mensagem[+] msg;

  @gmf.compartment(foo="bar")
  val LinhaTemporal[+] linhaTmp;

  @gmf.compartment(foo="bar")
  val ModulosTempo[+] ModTmp;
}

@gmf.node(label="cond", tool.name="Fragment Condition",
tool.description="Add a Fragment Condition to a Fragment. Not for use in
Fragments of Type: DivFrag and DiagSeq.")
class Condicao {
  attr String cond;
}

@gmf.link(label="tipo, nomeMsg", label.pattern="{0}: {1}", source="msgIn",
target="msgOut", style="solid", width="2", target.decoration="arrow",
tool.name="Message", tool.description="Add a Message to the Diagram. This
Message can be a MSG - Message or RTN - Return.")
class Mensagem {
  attr TipoMsg tipo;
  attr String nomeMsg;
  ref ModulosTempo[1] msgIn;
  ref ModulosTempo[1] msgOut;
}

@gmf.node(label="objecto")
abstract class Objectos {
  attr String objecto;

  @gmf.link(source="lig", target="lig", style="dot", width="2",
tool.small.bundle="Padrao.edit",
tool.small.path="/icons/full/obj16/LigObjModTmp.gif",
tool.large.bundle="Padrao.edit",
tool.large.path="/icons/full/obj16/LigObjModTmp.gif", tool.name="Link
Between an Object and a Time Module", tool.description="Add a Link Between
an Object and a Time Module to the Diagram")
  ref ModulosTempo[1] lig;
}

@gmf.node(figure="figures.ActorFigure", color="255,191,223",
border.color="193,0,97", tool.name="Actor", tool.description="Add an Actor
to the Diagram")
class Actor extends Objectos {
}

@gmf.node(figure="rectangle", color="255,200,145",
border.color="202,101,0", tool.name="Interface", tool.description="Add an
Interface to the Diagram")
class Interface extends Objectos {
}

@gmf.node(figure="rectangle", color="140,235,240",
border.color="14,108,114", tool.name="Control", tool.description="Add a
Control to the Diagram")
class Controlo extends Objectos {
}
```

```
@gmf.node(figure="rectangle", color="0,255,128", border.color="0,128,64",
tool.name="Entity", tool.description="Add an Entity to the Diagram")
class Entidade extends Objectos {
}

@gmf.link(source="tmpIn", target="tmpOut", style="dot", width="2",
tool.name="Time Line", tool.description="Add a Time Line to the Diagram.
Link 2 Time Modules.")
class LinhaTemporal {
  attr int linhaTmp;
  ref ModulosTempo[1] tmpIn;
  ref ModulosTempo[1] tmpOut;
}

@gmf.node(figure="rectangle", size="15,35", tool.name="Time Modules",
tool.description="Add a Time Module to the Diagram")
class ModulosTempo {
  attr String tmp;
}

enum TipoFrag {
  DiagSeq = 6;
  ALT = 0;
  OPT = 1;
  BREAK = 2;
  LOOP = 3;
  PAR = 4;
  DivFrag = 5;
}

enum TipoMsg {
  MSG = 0;
  RTN = 1;
}

@gmf.node(figure="rectangle", label="diagFeatures", tool.name="Features
Diagram", tool.description="Add a Features Diagram to the Pattern. Double
click to open a new editor.")
class DiagFeatures {
  attr String diagFeatures;

  @gmf.compartment(foo="bar")
  val Features[+] features;

  @gmf.compartment(foo="bar")
  val BinariosFeatures[+] binariosLig;

  @gmf.compartment(foo="bar")
  val DependenciasFeatures[+] dependenciasLig;

  @gmf.compartment(foo="bar")
  val Alternativa[+] altLig;

  @gmf.compartment(foo="bar")
  val AlternativaOpcional[+] altOpLig;

  @gmf.compartment(foo="bar")
  val Or[+] orLig;

  @gmf.compartment(foo="bar")
  val OrOpcional[+] orOpLig;
```

```
}

@gmf.node(figure="rounded", label="nomeFeature", tool.name="Feature",
tool.description="Add a Feature to the Diagram")
class Features {
  ref Features[+] ligacaoFeature;
  attr String nomeFeature;
}

@gmf.link(source="binLigIn", target="binLigOut")
abstract class BinariosFeatures {
  attr String nomeBin;
  ref Features[*] binLigIn;
  ref Features[*] binLigOut;
}

@gmf.link(style="solid", width="2", target.decoration="rhomb",
tool.name="Optional Link", tool.description="Add an Optional Link Between a
Feature 'mother' and a Feature 'daughter'")
class Opcional extends BinariosFeatures {
}

@gmf.link(style="solid", width="2", target.decoration="filledrhomb",
tool.name="Mandatory Link", tool.description="Add a Mandatory Link Between
a Feature 'mother' and a Feature 'daughter'")
class Obrigatorio extends BinariosFeatures {
}

@gmf.link(source="depLigIn", target="depLigOut")
abstract class DependenciasFeatures {
  attr String nomeDep;
  ref Features[*] depLigIn;
  ref Features[*] depLigOut;
}

@gmf.link(label="nomeDep", label.pattern="«exclude»{0}", style="dot",
width="2", color="145,145,145", tool.name="Exclude", tool.description="Add
an 'exclude' Relation Between 2 Features")
class Exclude extends DependenciasFeatures {
}

@gmf.link(label="nomeDep", label.pattern="«requires»{0}", style="dash",
width="2", color="145,145,145", tool.name="Requires", tool.description="Add
a 'requires' Relation Between 2 Features")
class Require extends DependenciasFeatures {
}

@gmf.link(source="AltLigObrIn", target="AltLigObrOut", style="solid",
width="1", color="204,102,0", tool.name="Alternative",
tool.description="Add an Alternative Relation Between Features of the Same
Level")
class Alternativa {
  ref Obrigatorio[*] AltLigObrIn;
  ref Obrigatorio[*] AltLigObrOut;
  attr String nomeAlternativa;
}

@gmf.link(source="AltOpLigOpIn", target="AltOpLigOpOut", style="solid",
width="1", color="204,102,0", tool.name="Optional Alternative",
tool.description="Add an Optional Alternative Relation Between Features of
the Same Level")
```

```
class AlternativaOpcional {
  ref Opcional[*] AltOpLigOpIn;
  ref Opcional[*] AltOpLigOpOut;
  attr String nomeAltOp;
}

@gmf.link(source="OrLigObrIn", target="OrLigObrOut", style="solid",
width="3", color="255,255,94", tool.name="Or", tool.description="Add an Or
Relation Between Features of the Same Level")
class Or {
  attr String nomeOr;
  ref Obrigatorio[*] OrLigObrIn;
  ref Obrigatorio[*] OrLigObrOut;
}

@gmf.link(source="OrOPLigOpIn", target="OrOpLigOpOut", style="solid",
width="3", color="255,255,94", tool.name="Optional Or",
tool.description="Add an Optional Or Relation Between Features of the Same
Level")
class OrOpcional {
  attr String nomeOrOp;
  ref Opcional[*] OrOPLigOpIn;
  ref Opcional[*] OrOpLigOpOut;
}
```

13. Anexo D – Metamodelo *emf* do Editor do Diagrama de *Features*

```
@namespace(uri="padrao", prefix="padrao")
package padrao;

class Padrao {
  val PadraoTemplate[1] Padrao;
}

class PadraoTemplate {
  attr String nomePadrao;
  val NomePadrao[1] nomePad;
  val ProblemaPadrao[1] problemaPad;
  val ContextoPadrao[1] contextoPad;
  val ReqsPadrao[1] PadReqs;
  val ListaEventosPadrao[1] listaEvtsPad;
  val Modelacao[1] modelacaoPad;
  val Consequencias[1] consequenciasPad;
  val Exemplos[1] exemplosPad;
  val PadroesRelacionados[1] PadroesRelacPad;
}

class NomePadrao {
  attr String nomeP;
}

class ContextoPadrao {
  attr String contextoP;
}

class ProblemaPadrao {
  attr String problemaP;
}

class ListaEventosPadrao {
  attr String listaEventosP;
  val Evento[*] evento;
}
class Evento {
  attr int ident = 1;
  attr String nomeEvento;
}

class ReqsPadrao {
  attr String nomeR;
  val Requisitos[2] ReqPad;
  val Funcionais[+] f;
  val NaoFuncionais[+] nf;
  val Dependencias[1] DepsPad;
```

```
  val Actores[1] ActsPad;
}

class Ligacao {
  ref Requisitos LigacaoIn;
  ref Requisitos[1] LigacaoOut;
}

abstract class Requisitos {
  attr String ident = "1";
  attr String descricao;
  val Ligacao[*] ligacoes;
}

class Funcionais extends Requisitos {
}

class NaoFuncionais extends Requisitos {
}

class Dependencias {
  val Requisitos[2..*] DepReq;
  attr String Id;
}

class Actores {
  attr String nome;
}

class Modelacao {
  val Estrutural[1] estruturalPad;
  attr String modelacao;
  val Comportamental[1] comportamentalPad;
}

class Estrutural {
  attr String estrutural;
  val DiagFeatures[1] dfeatures;
  val DiagClasses[1] dclasses;
}

class Comportamental {
  attr String comportamental;
  val DiagSequencia[1] dsequencia;
}

class Consequencias {
  attr String consequenciasP;
}

class Exemplos {
  attr String exemplosP;
  val DiagFeatures[1] dFeatureEx;
  val DiagClasses[1] dClasseEx;
  val DiagSequencia[1] dSequenciaEx;
  val Funcionais[1] ReqsEx;
}

class PadroesRelacionados {
  attr String padroesRelacionadosP;
  val PadraoRel[*] padRel;
```

```
}

class PadraoRel {
  attr int idPadrao = 1;
  attr String nomePadraoRel;
}

class DiagClasses {
  val Classe[+] classeDC;
  attr String diagClasses;
}

class Classe {
  val Atributo[+] atribDC;
  val Metodo[+] metodoDC;
  val Dependencia[+] dependenciaDC;
  val Generalizacao[+] generalizacaoDC;
  attr String classe;
}

@gmf.compartment(layout="list")
class Atributo {
  attr String atrib;
  val TipoAtributo[1] tipoAtributo;
}

class TipoAtributo {
  attr String tipoAtrib;
}

class Metodo {
  attr String metodo;
  val TipoRetornoMet[1] tipoRetornoMetodo;
  val ParametrosMetodo[+] parametrosMetodo;
}

class TipoRetornoMet {
  attr String tipoRetornoMet;
}

class ParametrosMetodo {
  attr String parametrosMet;
  val TipoParametro[1] tipoParam;
}

class TipoParametro {
  attr String tipoParam;
}

abstract class Dependencia {
  ref Classe[1] dependenciaDCout;
  attr String cardinalidadeIn;
  attr String cardinalidadeOut;
  ref Classe[1] dependenciaDCin;
}

class Agregacao extends Dependencia {
}

class Composicao extends Dependencia {
}
```

```
class Generalizacao {
  ref Classe[1] generalizacaoIn;
  ref Classe[1] generalizacaoOut;
}

class DiagSequencia {
  attr String diagSequencia;
  val Fragmentos[+] elems;
}

class Fragmentos {
  attr TipoFrag tipoFrag;
  val Condicao condicaoFrag;
  val Fragmentos[*] frags;
  val Objectos[*] obj;
  val Mensagem[+] msg;
  val LinhaTemporal[+] linhaTmp;
  val ModulosTempo[+] ModTmp;
}

class Condicao {
  attr String cond;
}

class Mensagem {
  attr TipoMsg tipo;
  attr String nomeMsg;
  ref ModulosTempo[1] msgIn;
  ref ModulosTempo[1] msgOut;
}

abstract class Objectos {
  attr String objecto;
  ref ModulosTempo[1] lig;
}

class Actor extends Objectos {
}

class Interface extends Objectos {
}

class Controlo extends Objectos {
}

class Entidade extends Objectos {
}

class LinhaTemporal {
  attr int linhaTmp;
  ref ModulosTempo[1] tmpIn;
  ref ModulosTempo[1] tmpOut;
}

class ModulosTempo {
  attr String tmp;
}

enum TipoFrag {
  DiagSeq = 6;
```

```
  ALT = 0;
  OPT = 1;
  BREAK = 2;
  LOOP = 3;
  PAR = 4;
  DivFrag = 5;
}

enum TipoMsg {
  MSG = 0;
  RTN = 1;
}

@gmf.diagram(foo="bar")
class DiagFeatures {
  attr String diagFeatures;
  val Features[+] features;
  val BinariosFeatures[+] binariosLig;
  val DependenciasFeatures[+] dependenciasLig;
  val Alternativa[+] altLig;
  val AlternativaOpcional[+] altOpLig;
  val Or[+] orLig;
  val OrOpcional[+] orOpLig;
}

@gmf.node(figure="rounded", label="nomeFeature", tool.name="Feature",
tool.description="Add a Feature to the Diagram")
class Features {
  ref Features[+] ligacaoFeature;
  attr String nomeFeature;
}

@gmf.link(source="binLigIn", target="binLigOut")
abstract class BinariosFeatures {
  attr String nomeBin;
  ref Features[*] binLigIn;
  ref Features[*] binLigOut;
}

@gmf.link(style="solid", width="2", target.decoration="rhomb",
tool.name="Optional Link", tool.description="Add an Optional Link Between a
Feature 'mother' and a Feature 'daughter'")
class Opcional extends BinariosFeatures {
}

@gmf.link(style="solid", width="2", target.decoration="filledrhomb",
tool.name="Mandatory Link", tool.description="Add a Mandatory Link Between
a Feature 'mother' and a Feature 'daughter'")
class Obrigatorio extends BinariosFeatures {
}

@gmf.link(source="depLigIn", target="depLigOut")
abstract class DependenciasFeatures {
  attr String nomeDep;
  ref Features[*] depLigIn;
  ref Features[*] depLigOut;
}

@gmf.link(label="nomeDep", label.pattern="«exclude»{0}", style="dot",
width="2", color="145,145,145", tool.name="Exclude", tool.description="Add
an 'exclude' Relation Between 2 Features")
```

```
class Exclude extends DependenciasFeatures {
}

@gmf.link(label="nomeDep", label.pattern="«requires»{0}", style="dash",
width="2", color="145,145,145", tool.name="Requires", tool.description="Add
a 'requires' Relation Between 2 Features")
class Require extends DependenciasFeatures {
}

@gmf.link(source="AltLigObrIn", target="AltLigObrOut", style="solid",
width="1", color="204,102,0", tool.name="Alternative",
tool.description="Add an Alternative Relation Between Features of the Same
Level")
class Alternativa {
  ref Obrigatorio[*] AltLigObrIn;
  ref Obrigatorio[*] AltLigObrOut;
  attr String nomeAlternativa;
}

@gmf.link(source="AltOpLigOpIn", target="AltOpLigOpOut", style="solid",
width="1", color="204,102,0", tool.name="Optional Alternative",
tool.description="Add an Optional Alternative Relation Between Features of
the Same Level")
class AlternativaOpcional {
  ref Opcional[*] AltOpLigOpIn;
  ref Opcional[*] AltOpLigOpOut;
  attr String nomeAltOp;
}

@gmf.link(source="OrLigObrIn", target="OrLigObrOut", style="solid",
width="3", color="255,255,94", tool.name="Or", tool.description="Add an Or
Relation Between Features of the Same Level")
class Or {
  attr String nomeOr;
  ref Obrigatorio[*] OrLigObrIn;
  ref Obrigatorio[*] OrLigObrOut;
}

@gmf.link(source="OrOPLigOpIn", target="OrOpLigOpOut", style="solid",
width="3", color="255,255,94", tool.name="Optional Or",
tool.description="Add an Optional Or Relation Between Features of the Same
Level")
class OrOpcional {
  attr String nomeOrOp;
  ref Opcional[*] OrOPLigOpIn;
  ref Opcional[*] OrOpLigOpOut;
}
```

14. Anexo E – Metamodelo *emf* do Editor do Diagrama de Classes

```
@namespace(uri="padrao", prefix="padrao")
package padrao;

class Padrao {
  val PadraoTemplate[1] Padrao;
}

class PadraoTemplate {
  attr String nomePadrao;
  val NomePadrao[1] nomePad;
  val ProblemaPadrao[1] problemaPad;
  val ContextoPadrao[1] contextoPad;
  val ReqsPadrao[1] PadReqs;
  val ListaEventosPadrao[1] listaEvtsPad;
  val Modelacao[1] modelacaoPad;
  val Consequencias[1] consequenciasPad;
  val Exemplos[1] exemplosPad;
  val PadroesRelacionados[1] PadroesRelacPad;
}

class NomePadrao {
  attr String nomeP;
}

class ContextoPadrao {
  attr String contextoP;
}

class ProblemaPadrao {
  attr String problemaP;
}

class ListaEventosPadrao {
  attr String listaEventosP;
  val Evento[*] evento;
}
class Evento {
  attr int ident = 1;
  attr String nomeEvento;
}

class ReqsPadrao {
  attr String nomeR;
  val Requisitos[2] ReqPad;
  val Funcionais[+] f;
  val NaoFuncionais[+] nf;
  val Dependencias[1] DepsPad;
```

```
  val Actores[1] ActsPad;
}

class Ligacao {
  ref Requisitos LigacaoIn;
  ref Requisitos[1] LigacaoOut;
}

abstract class Requisitos {
  attr String ident = "1";
  attr String descricao;
  val Ligacao[*] ligacoes;
}

class Funcionais extends Requisitos {
}

class NaoFuncionais extends Requisitos {
}

class Dependencias {
  val Requisitos[2..*] DepReq;
  attr String Id;
}

class Actores {
  attr String nome;
}

class Modelacao {
  val Estrutural[1] estruturalPad;
  attr String modelacao;
  val Comportamental[1] comportamentalPad;
}

class Estrutural {
  attr String estrutural;
  val DiagFeatures[1] dfeatures;
  val DiagClasses[1] dclasses;
}

class Comportamental {
  attr String comportamental;
  val DiagSequencia[1] dsequencia;
}

class Consequencias {
  attr String consequenciasP;
}

class Exemplos {
  attr String exemplosP;
  val DiagFeatures[1] dFeatureEx;
  val DiagClasses[1] dClasseEx;
  val DiagSequencia[1] dSequenciaEx;
  val Funcionais[1] ReqsEx;
}

class PadroesRelacionados {
  attr String padroesRelacionadosP;
  val PadraoRel[*] padRel;
```

```
}

class PadraoRel {
  attr int idPadrao = 1;
  attr String nomePadraoRel;
}

@gmf.diagram(foo="bar")
class DiagClasses {

  @gmf.compartment(foo="bar")
  val Classe[+] classeDC;
  attr String diagClasses;
}

@gmf.node(figure="rounded", label="classe", tool.name="Class",
tool.description="Add a new Class to the Diagram")
class Classe {

  @gmf.compartment(layout="list")
  val Atributo[+] atribDC;

  @gmf.compartment(layout="list")
  val Metodo[+] metodoDC;
  val Dependencia[+] dependenciaDC;
  val Generalizacao[+] generalizacaoDC;
  attr String classe;
}

@gmf.node(figure="rectangle", label="atrib", tool.name="Attribute",
tool.description="Add a new Attribute to the Class")
@gmf.compartment(layout="list")
class Atributo {
  attr String atrib;

  @gmf.compartment(layout="list")
  val TipoAtributo[1] tipoAtributo;
}

@gmf.node(figure="rectangle", label="tipoAtrib", tool.name="Attribute
Type", tool.description="Add the Attribute Type to the Attribute")
class TipoAtributo {
  attr String tipoAtrib;
}

@gmf.node(figure="rectangle", label="metodo", tool.name="Method",
tool.description="Add a new Method to the Class")
@gmf.compartment(layout="list")
class Metodo {
  attr String metodo;

  @gmf.compartment(layout="list")
  val TipoRetornoMet[1] tipoRetornoMetodo;

  @gmf.compartment(layout="list")
  val ParametrosMetodo[+] parametrosMetodo;
}

@gmf.node(figure="rectangle", label="tipoRetornoMet", tool.name="Method's
Return Type", tool.description="Add the Method's Return Type to the
Method")
```

```
class TipoRetornoMet {
  attr String tipoRetornoMet;
}

@gmf.node(figure="rectangle", label="parametrosMet", tool.name="Method's
Parameter", tool.description="Add a Method's Parameter")
class ParametrosMetodo {
  attr String parametrosMet;

  @gmf.compartment(layout="list")
  val TipoParametro[1] tipoParam;
}

@gmf.node(figure="rectangle", label="tipoParam", tool.name="Parameter's
Type", tool.description="Add a Parameter's Type to the Parameter")
class TipoParametro {
  attr String tipoParam;
}

@gmf.link(label="cardinalidadeIn, cardinalidadeOut", label.pattern="{0}
,           {1}", source="dependenciaDCin", target="dependenciaDCout")
abstract class Dependencia {
  ref Classe[1] dependenciaDCout;
  attr String cardinalidadeIn;
  attr String cardinalidadeOut;
  ref Classe[1] dependenciaDCin;
}

@gmf.link(style="dot", width="2", color="56,114,114",
target.decoration="filledrhomb", tool.name="Aggregation",
tool.description="Add an Aggregation Relation Between two Classes")
class Agregacao extends Dependencia {
}
@gmf.link(style="dot", width="2", color="159,0,159",
target.decoration="arrow", tool.name="Composition", tool.description="Add a
Composition Relation Between two Classes")
class Composicao extends Dependencia {
}

@gmf.link(source="generalizacaoIn", target="generalizacaoOut",
style="solid", width="1", color="0,0,174", target.decoration="closedarrow",
tool.name="Generalization", tool.description="Add a Generalization Relation
Between two Classes")
class Generalizacao {
  ref Classe[1] generalizacaoIn;
  ref Classe[1] generalizacaoOut;
}

class DiagSequencia {
  attr String diagSequencia;
  val Fragmentos[+] elems;
}

class Fragmentos {
  attr TipoFrag tipoFrag;
  val Condicao condicaoFrag;
  val Fragmentos[*] frags;
  val Objectos[*] obj;
  val Mensagem[+] msg;
  val LinhaTemporal[+] linhaTmp;
  val ModulosTempo[+] ModTmp;
```

```
}

class Condicao {
  attr String cond;
}

class Mensagem {
  attr TipoMsg tipo;
  attr String nomeMsg;
  ref ModulosTempo[1] msgIn;
  ref ModulosTempo[1] msgOut;
}

abstract class Objectos {
  attr String objecto;
  ref ModulosTempo[1] lig;
}

class Actor extends Objectos {
}

class Interface extends Objectos {
}

class Controlo extends Objectos {
}

class Entidade extends Objectos {
}

class LinhaTemporal {
  attr int linhaTmp;
  ref ModulosTempo[1] tmpIn;
  ref ModulosTempo[1] tmpOut;
}

class ModulosTempo {
  attr String tmp;
}

enum TipoFrag {
  DiagSeq = 6;
  ALT = 0;
  OPT = 1;
  BREAK = 2;
  LOOP = 3;
  PAR = 4;
  DivFrag = 5;
}

enum TipoMsg {
  MSG = 0;
  RTN = 1;
}

class DiagFeatures {
  attr String diagFeatures;
  val Features[+] features;
  val BinariosFeatures[+] binariosLig;
  val DependenciasFeatures[+] dependenciasLig;
  val Alternativa[+] altLig;
```

```
  val AlternativaOpcional[+] altOpLig;
  val Or[+] orLig;
  val OrOpcional[+] orOpLig;
}

class Features {
  ref Features[+] ligacaoFeature;
  attr String nomeFeature;
}

abstract class BinariosFeatures {
  attr String nomeBin;
  ref Features[*] binLigIn;
  ref Features[*] binLigOut;
}

class Opcional extends BinariosFeatures {
}

class Obrigatorio extends BinariosFeatures {
}
abstract class DependenciasFeatures {
  attr String nomeDep;
  ref Features[*] depLigIn;
  ref Features[*] depLigOut;
}

class Exclude extends DependenciasFeatures {
}

class Require extends DependenciasFeatures {
}

class Alternativa {
  ref Obrigatorio[*] AltLigObrIn;
  ref Obrigatorio[*] AltLigObrOut;
  attr String nomeAlternativa;
}

class AlternativaOpcional {
  ref Opcional[*] AltOpLigOpIn;
  ref Opcional[*] AltOpLigOpOut;
  attr String nomeAltOp;
}

class Or {
  attr String nomeOr;
  ref Obrigatorio[*] OrLigObrIn;
  ref Obrigatorio[*] OrLigObrOut;
}

class OrOpcional {
  attr String nomeOrOp;
  ref Opcional[*] OrOPLigOpIn;
  ref Opcional[*] OrOpLigOpOut;
}
```

15. Anexo F – Metamodelo *emf* do Editor do Diagrama de Sequência

```
@namespace(uri="padrao", prefix="padrao")
package padrao;

class Padrao {
  val PadraoTemplate[1] Padrao;
}

class PadraoTemplate {
  attr String nomePadrao;
  val NomePadrao[1] nomePad;
  val ProblemaPadrao[1] problemaPad;
  val ContextoPadrao[1] contextoPad;
  val ReqsPadrao[1] PadReqs;
  val ListaEventosPadrao[1] listaEvtsPad;
  val Modelacao[1] modelacaoPad;
  val Consequencias[1] consequenciasPad;
  val Exemplos[1] exemplosPad;
  val PadroesRelacionados[1] PadroesRelacPad;
}

class NomePadrao {
  attr String nomeP;
}

class ContextoPadrao {
  attr String contextoP;
}

class ProblemaPadrao {
  attr String problemaP;
}

class ListaEventosPadrao {
  attr String listaEventosP;
  val Evento[*] evento;
}
class Evento {
  attr int ident = 1;
  attr String nomeEvento;
}

class ReqsPadrao {
  attr String nomeR;
  val Requisitos[2] ReqPad;
  val Funcionais[+] f;
  val NaoFuncionais[+] nf;
  val Dependencias[1] DepsPad;
```

```
  val Actores[1] ActsPad;
}

class Ligacao {
  ref Requisitos LigacaoIn;
  ref Requisitos[1] LigacaoOut;
}

abstract class Requisitos {
  attr String ident = "1";
  attr String descricao;
  val Ligacao[*] ligacoes;
}

class Funcionais extends Requisitos {
}

class NaoFuncionais extends Requisitos {
}

class Dependencias {
  val Requisitos[2..*] DepReq;
  attr String Id;
}

class Actores {
  attr String nome;
}

class Modelacao {
  val Estrutural[1] estruturalPad;
  attr String modelacao;
  val Comportamental[1] comportamentalPad;
}

class Estrutural {
  attr String estrutural;
  val DiagFeatures[1] dfeatures;
  val DiagClasses[1] dclasses;
}

class Comportamental {
  attr String comportamental;
  val DiagSequencia[1] dsequencia;
}

class Consequencias {
  attr String consequenciasP;
}

class Exemplos {
  attr String exemplosP;
  val DiagFeatures[1] dFeatureEx;
  val DiagClasses[1] dClasseEx;
  val DiagSequencia[1] dSequenciaEx;
  val Funcionais[1] ReqsEx;
}

class PadroesRelacionados {
  attr String padroesRelacionadosP;
  val PadraoRel[*] padRel;
```

```
}

class PadraoRel {
  attr int idPadrao = 1;
  attr String nomePadraoRel;
}

class DiagClasses {
  val Classe[+] classeDC;
  attr String diagClasses;
}

class Classe {
  val Atributo[+] atribDC;
  val Metodo[+] metodoDC;
  val Dependencia[+] dependenciaDC;
  val Generalizacao[+] generalizacaoDC;
  attr String classe;
}

class Atributo {
  attr String atrib;
  val TipoAtributo[1] tipoAtributo;
}

class TipoAtributo {
  attr String tipoAtrib;
}

class Metodo {
  attr String metodo;
  val TipoRetornoMet[1] tipoRetornoMetodo;
  val ParametrosMetodo[+] parametrosMetodo;
}

class TipoRetornoMet {
  attr String tipoRetornoMet;
}
class ParametrosMetodo {
  attr String parametrosMet;
  val TipoParametro[1] tipoParam;
}

class TipoParametro {
  attr String tipoParam;
}

abstract class Dependencia {
  ref Classe[1] dependenciaDCout;
  attr String cardinalidadeIn;
  attr String cardinalidadeOut;
  ref Classe[1] dependenciaDCin;
}

class Agregacao extends Dependencia {
}

class Composicao extends Dependencia {
}

class Generalizacao {
```

```
  ref Classe[1] generalizacaoIn;
  ref Classe[1] generalizacaoOut;
}

@gmf.diagram(foo="bar")
class DiagSequencia {
  attr String diagSequencia;

  @gmf.compartment(foo="bar")
  val Fragmentos[+] elems;
}

@gmf.node(label="tipoFrag", tool.name="Fragment", tool.description="Add a
Fragment to the Diagram. These fragments can be of type: Sequence Diagram,
ALT, OPT, LOOP, PAR, BREAK e DivFrag, the last one is used to divide
fragments of type ALT and PAR.")
class Fragmentos {
  attr TipoFrag tipoFrag;

  @gmf.compartment(foo="bar")
  val Condicao condicaoFrag;

  @gmf.compartment(foo="bar")
  val Fragmentos[*] frags;

  @gmf.compartment(foo="bar")
  val Objectos[*] obj;
  val Mensagem[+] msg;

  @gmf.compartment(foo="bar")
  val LinhaTemporal[+] linhaTmp;

  @gmf.compartment(foo="bar")
  val ModulosTempo[+] ModTmp;
}

@gmf.node(label="cond", tool.name="Fragment Condition",
tool.description="Add a Fragment Condition to a Fragment. Not for use in
Fragments of Type: DivFrag and DiagSeq.")
class Condicao {
  attr String cond;
}

@gmf.link(label="tipo, nomeMsg", label.pattern="{0}: {1}", source="msgIn",
target="msgOut", style="solid", width="2", target.decoration="arrow",
tool.name="Message", tool.description="Add a Message to the Diagram. This
Message can be a MSG - Message or RTN - Return.")
class Mensagem {
  attr TipoMsg tipo;
  attr String nomeMsg;
  ref ModulosTempo[1] msgIn;
  ref ModulosTempo[1] msgOut;
}

@gmf.node(label="objecto")
abstract class Objectos {
  attr String objecto;

  @gmf.link(source="lig", target="lig", style="dot", width="2",
tool.small.bundle="Pattern.edit",
tool.small.path="/icons/full/obj16/LigObjModTmp.gif",
```

```
tool.large.bundle="Pattern.edit",
tool.large.path="/icons/full/obj16/LigObjModTmp.gif", tool.name="Link
Between an Object and a Time Module", tool.description="Add a Link Between
an Object and a Time Module to the Diagram")
  ref ModulosTempo[1] lig;
}

@gmf.node(figure="figures.ActorFigure", color="255,191,223",
border.color="193,0,97", tool.name="Actor", tool.description="Add an Actor
to the Diagram")
class Actor extends Objectos {
}

@gmf.node(figure="rectangle", color="255,200,145",
border.color="202,101,0", tool.name="Interface", tool.description="Add an
Interface to the Diagram")
class Interface extends Objectos {
}

@gmf.node(figure="rectangle", color="140,235,240",
border.color="14,108,114", tool.name="Control", tool.description="Add a
Control to the Diagram")
class Controlo extends Objectos {
}

@gmf.node(figure="rectangle", color="0,255,128", border.color="0,128,64",
tool.name="Entity", tool.description="Add an Entity to the Diagram")
class Entidade extends Objectos {
}

@gmf.link(source="tmpIn", target="tmpOut", style="dot", width="2",
tool.name="Time Line", tool.description="Add a Time Line to the Diagram.
Link 2 Time Modules.")
class LinhaTemporal {
  attr int linhaTmp;
  ref ModulosTempo[1] tmpIn;
  ref ModulosTempo[1] tmpOut;
}

@gmf.node(figure="rectangle", size="15,35", tool.name="Time Modules",
tool.description="Add a Time Module to the Diagram")
class ModulosTempo {
  attr String tmp;
}

enum TipoFrag {
  DiagSeq = 6;
  ALT = 0;
  OPT = 1;
  BREAK = 2;
  LOOP = 3;
  PAR = 4;
  DivFrag = 5;
}

enum TipoMsg {
  MSG = 0;
  RTN = 1;
}

class DiagFeatures {
```

```
  attr String diagFeatures;
  val Features[+] features;
  val BinariosFeatures[+] binariosLig;
  val DependenciasFeatures[+] dependenciasLig;
  val Alternativa[+] altLig;
  val AlternativaOpcional[+] altOpLig;
  val Or[+] orLig;
  val OrOpcional[+] orOpLig;
}

class Features {
  ref Features[+] ligacaoFeature;
  attr String nomeFeature;
}

abstract class BinariosFeatures {
  attr String nomeBin;
  ref Features[*] binLigIn;
  ref Features[*] binLigOut;
}

class Opcional extends BinariosFeatures {
}

class Obrigatorio extends BinariosFeatures {
}

abstract class DependenciasFeatures {
  attr String nomeDep;
  ref Features[*] depLigIn;
  ref Features[*] depLigOut;
}

class Exclude extends DependenciasFeatures {
}

class Require extends DependenciasFeatures {
}

class Alternativa {
  ref Obrigatorio[*] AltLigObrIn;
  ref Obrigatorio[*] AltLigObrOut;
  attr String nomeAlternativa;
}

class AlternativaOpcional {
  ref Opcional[*] AltOpLigOpIn;
  ref Opcional[*] AltOpLigOpOut;
  attr String nomeAltOp;
}

class Or {
  attr String nomeOr;
  ref Obrigatorio[*] OrLigObrIn;
  ref Obrigatorio[*] OrLigObrOut;
}

class OrOpcional {
  attr String nomeOrOp;
  ref Opcional[*] OrOPLigOpIn;
  ref Opcional[*] OrOpLigOpOut;
```

```
}
```

16. Anexo G – Manual de Utilização da Ferramenta *PatternTool*

Esta ferramenta tem como objectivo permitir a descrição de padrões de análise através do preenchimento de um *template* de padrão de análise.

Para poder usar este *plugin* no Eclipse[10] é necessário colocar os 8 ficheiros *.jar, na pasta *plugins* do seu eclipse. Feito isto apenas tem de abrir o Eclipse e fazer *File>New>Project* e escolher *General>Project*. Depois de criado o projecto basta fazer *File>New>Example* e Escolher *Padrao Diagram*.

Assim, quando se inicia o *plugin* é apresentada uma área de edição e uma palete à esquerda onde estão todos os campos que podem ser adicionados. Os campos foram organizados pela ordem em que devem ser colocados no *template* e as cores indicam a hierarquia da colocação. Isto é, a cor laranja indica os elementos raiz, dentro dos compartimentos dos elementos de *icon* laranja são colocados os elementos de *icon* verde-escuro. Dentro dos compartimentos dos elementos com *icon* verde-escuro colocam-se os elementos de *icon* verde-claro e, finalmente, dentro dos compartimentos de *icon* verde-claro colocam-se os elementos de *icon* verde-mar.

Assim, a criação de um *template* inicia-se com a colocação de um elemento *Pattern Template*. Depois de colocar este elemento, clicar nele com o botão direito do rato e seleccionar a opção *Show Properties View*, surge na parte inferior uma janela onde irão surgindo, ao longo da edição, os campos editáveis e é aqui que estes devem ser editados.

Dentro do elemento *Pattern Template* surgem diversos compartimentos, em cada um desses compartimentos vamos colocar outros elementos. Isto é, a estrutura em árvore do template deverá ser a seguinte:

1. *Pattern's Name*;
2. *Pattern's Context*;
3. *Pattern's Problem*;
4. *Pattern's Requirements Analysis*:
    a. *Functional Requirement*;
    b. *Non-Functional Requirement*;
    c. *Dependencies Between Requirements*:

[10] Este plugin foi testado no Eclipse Galileu Modelling.

i. *Functional Requirement*;

ii. *Non-Functional Requirement*;

iii. *Dependency Connection Between Requirements* (para ligar os requisitos Funcionais aos Requisitos Não-Funcionais);

5. *Pattern's Events List*:
   a. *Pattern's Event*;
6. *Modelation*:
   a. *Structural Modelation*:
      i. *Features Diagram*;
      ii. *Class Diagram*;
   b. Behavioral Modelation:
      i. *Sequence Diagram*;
7. *Pattern's Application Consequences*;
8. *Examples*:
   a. *Functional Requirement*;
   b. *Features Diagram*;
   c. *Class Diagram*;
   d. *Sequence Diagram*;
9. *Related Patterns Section*:
   a. *Related Pattern*.

Para criar as dependências entre os requisitos colocar dentro do compartimento das dependências os requisitos funcionais e não funcionais e ligá-los com a ligação *Dependency Connection Between Requirements*.

Para editar os diagramas de classes, *features* e de sequência tem de se clicar duas vezes com o botão esquerdo do rato nas bordas do compartimento, isto irá abrir um sub-editor do diagrama correspondente. Depois de criado o diagrama é necessário salvar o ficheiro e, assim aparecerá no editor principal o diagrama criado no sub-editor. Os diagramas irão aparecer no editor principal com todos os elementos sobrepostos, pois a *framework* não guarda as posições de um editor para outro. No entanto, pode atenuar-se esta apresentação seleccionando o compartimento do diagrama em causa, dentro do compartimento que surge no editor principal e na barra de tarefas do *eclipse* escolher a opção *Arrange All*.

Relativamente aos exemplos, podemos ter vários exemplos, devendo colocar-se um compartimento por exemplo. Dentro de um exemplo vamos ter um conjunto de requisitos

funcionais, um diagrama de *features*, um diagrama de classes e de um a três diagramas de sequência.

Diagrama de *Features*

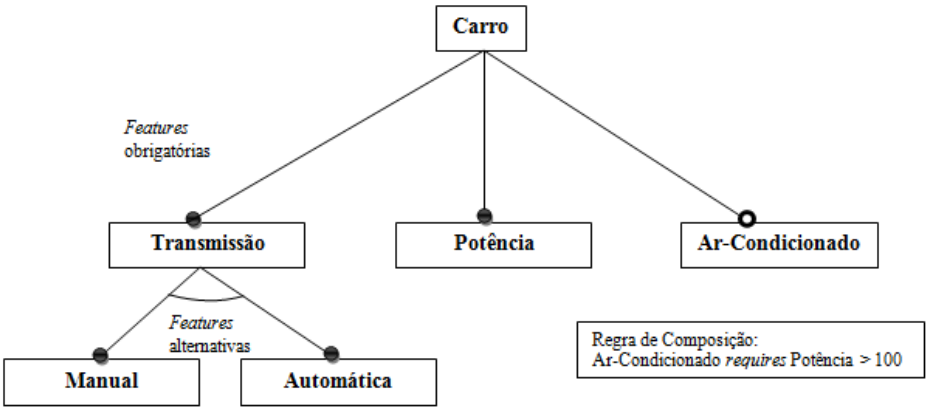

Figura 16.1. Exemplo de Modelo de *Features*[11]

Um modelo de *features* (Figura 16.1) representa as *features standard* de uma família de sistemas no domínio e as relações entre elas. Cada *feature* do modelo deve ter um nome diferente dos das restantes *features*.

As *features* opcionais são *features* que podem ou não ser seleccionadas – ○;

As *features* obrigatórias são *features* que têm de ser, obrigatoriamente, seleccionadas – ●;

Um grupo de *features* é:

1. "Or Opcional" se as *features* forem opcionais e apenas uma delas poder ser seleccionada de cada vez – ;
2. "Or" se as *features* forem obrigatórias e tiver de se seleccionar apenas uma de cada vez – ;
3. "Alternative" se as *features* forem obrigatórias e se poder seleccionar mais do que uma *feature* de cada vez – ;
4. "Optional Alternative" se as *features* forem opcionais e se poder seleccionar mais do que uma *feature* de cada vez – ;

Para construir o diagrama de *features* deve começar-se por colocar a *feature* raiz e no nível abaixo as suas filhas e por aí em diante. As ligações entre as *features* devem ser feitas, sempre da *feature* mãe para a *feature* filha. Estas ligações são as *Optional Link* e *Mandatory Link*. As restantes relações, *Alternative*, *Optional Alternative*, *Or* e *Optional Or* são colocadas entre as ligações de *Optional Link* e *Mandatory Link*.

[11] **Kang, K.**, Cohen, S., Hess, J., Novak, W., Peterson, A., *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Domain Analysis Project, Software Engineering Institute, Carnegie Mellon University Pittsburgh, Pennsylvania, Novembro 1990.

Diagrama de Classes

Para criar um diagrama de classes colocam-se tantas classes quanto as necessárias. Dentro dessas classes colocamos os atributos necessários e dentro de cada atributo colocamos o tipo desse atribuí. Dentro das classes colocamos também, os métodos das classes e dentro desses métodos colocamos o tipo de retorno dos métodos e os parâmetros necessários. Dentro dos parâmetros colocamos o tipo dos parâmetros.

Podem usar-se as ligações de composição, agregação e generalização entre as classes. Nas ligações de composição e agregação podem atribuir-se cardinalidades.

Diagrama de Sequência

Para construir o Diagrama de Sequência começar por colocar um Fragmento, este deve ser do tipo DiagSeq. Colocam-se os objectos necessários, sendo que estes podem ser do tipo Utilizador, Interface, Controlo ou Entidade. Por baixo de cada objecto colocam-se módulos de tempo, o primeiro módulo é ligado ao objecto através da ligação *Link Between an Object and a Time Module*, os restantes módulos de tempo ligam-se entre si através das ligações *Time Line*. As mensagens entre os objectos devem ser colocadas de módulo de tempo para módulo de tempo.

Para inserir fragmentos do tipo ALT, OPT, BREAK, PAR e LOOP, colocar um fragmento e escolher o tipo desejado. Para colocar as divisões nos fragmentos colocar um novo fragmento dentro dele e escolher o tipo DivFrag. As condições dos fragmentos são colocadas dentro de um fragmento e através do elemento *Fragment Condition*.

Nota: Se for necessário apagar algum elemento, seleccione esse elemento, carregue com o botão direito do rato sobre ele e seleccione a opção *Delete From Model.*